Coupling Large Language Models with Logic Programming for Robust and General Reasoning from Text

Zhun Yang, Adam Ishay [1]
[1] Arizona State University
{zyang90,aishay}@asu.edu

Joohyung Lee [1,2]
[2] Samsung Research
joolee@asu.edu

Abstract

While large language models (LLMs), such as GPT-3, appear to be robust and general, their reasoning ability is not at a level to compete with the best models trained for specific natural language reasoning problems. In this study, we observe that a large language model can serve as a highly effective few-shot semantic parser. It can convert natural language sentences into a logical form that serves as input for answer set programs, a logic-based declarative knowledge representation formalism. The combination results in a robust and general system that can handle multiple question-answering tasks without requiring retraining for each new task. It only needs a few examples to guide the LLM's adaptation to a specific task, along with reusable ASP knowledge modules that can be applied to multiple tasks. We demonstrate that this method achieves state-of-the-art performance on several NLP benchmarks, including bAbI, StepGame, CLUTRR, and gSCAN. Additionally, it successfully tackles robot planning tasks that an LLM alone fails to solve.

1 Introduction

A typical way to handle a question-answering task is to train a neural network model on large training data and test it on similar data. Such models work well with linguistic variability and ambiguity but often learn statistical features and correlations rather than true reasoning (Ruder, 2021), which makes them not robust, lack generalization, and difficult to interpret.

Alternatively, transformer-based large language models (LLMs) have recently shown wide success on many downstream tasks, demonstrating general reasoning capability on diverse tasks without being retrained. However, when we restrict our attention to individual NLP reasoning benchmarks, they usually do not perform as well as state-of-the-art models despite various efforts to improve accuracy through prompt engineering (Wei et al., 2022; Zhou et al., 2022).

Similarly, LLMs gained attention for plan generation for robots due to the rich semantic knowledge they acquired about the world (Ahn et al., 2022; Huang et al., 2022; Zeng et al., 2022). However, LLMs are known to perform shallow reasoning and cannot find complex plans (Valmeekam et al., 2022).

In another context, Nye et al. (2021) note that while LLMs are good at System-1 thinking, their outputs are often inconsistent and incoherent. This is because LLMs are trained to predict subsequent words in a sequence and do not appear to have a deep understanding of concepts such as cause and effect, logic, and probability, which are important for reasoning.

Nevertheless, we note that the rich semantic knowledge that LLMs possess makes them effective general-purpose few-shot semantic parsers that can convert linguistically variable natural language sentences into atomic facts that serve as input to logic programs. We also note that the fully declarative nature of answer set programs (Lifschitz, 2008; Brewka et al., 2011) makes them a good pair with the LLM semantic parsers, providing interpretable and explainable reasoning on the parsed result of the LLMs using background knowledge. Combining large language models and answer set programs leads to an attractive dual-process, neuro-symbolic reasoning that works across multiple QA tasks without retraining for individual tasks.

We tested this idea with several NLP benchmarks, bAbI (Weston et al., 2016), StepGame (Shi et al., 2022), CLUTRR (Sinha et al., 2019), and gSCAN (Ruis et al., 2020), by applying the same dual-system model and achieved state-of-the-art performance in all of them. Furthermore, the high accuracy and transparency allow us to easily identify the source of errors, making our system a useful data set validation tool as well. In particular, we found a significant amount of errors in the original

Findings of the Association for Computational Linguistics: ACL 2023, pages 5186–5219
July 9-14, 2023 ©2023 Association for Computational Linguistics

CLUTRR dataset that are hard to detect manually.

While the new version of GPT-3 (Brown et al., 2020) (text-davinci-003) shows improvement over its predecessors, we observe that it also retains critical limitations. In the process, we develop prompt methods for semantic parsing to overcome some of them.

The implementation of our method is publicly available online at `https://github.com/azreasoners/LLM-ASP`.

2 Preliminaries

2.1 Semantic Parsing and LLMs

Semantic parsing involves converting a natural language query or statement into a structured representation that a computer can understand and manipulate. Statistical methods have increased in popularity (Zelle and Mooney, 1996; Miller et al., 1996; Zettlemoyer and Collins, 2005; Wong and Mooney, 2007), and encoder-decoder models in particular have been widely used (Dong and Lapata, 2016; Jia and Liang, 2016; Kočiskỳ et al., 2016). However, these statistical methods require annotated input and output pairs. Furthermore, machine learning models often fail to compositionally generalize to unseen data (Lake and Baroni, 2018).

More recently, pre-trained language models have been applied to semantic parsing tasks (Liu et al., 2021), such as generating SQL queries, SPARQL queries, logical forms, or programs, from natural language, together with fine-tuning or prompt-tuning on pre-trained models, such as BART, RoBERTa and GPT-2 (Chen et al., 2020a; Shin et al., 2021; Schucher et al., 2022). With larger pre-trained networks, such as GPT-3, prompting appears to yield a reasonable semantic parser without the need for fine-tuning (Shin et al., 2021; Drozdov et al., 2022).

Another line of related work is to apply pre-trained language models to relation extraction, the task of extracting semantic relationships from a text given two or more entities (Liu et al., 2021). Wang et al. (2022) do zero-shot relation extraction with pre-trained language models from the BERT family and GPT-2 variants. Zhou and Chen (2022) fine-tune BERT and RoBERTa models for the extraction of sentence-level relations. Chen et al. (2022) apply prompt-tuning to RoBERT_LARGE for relation extraction. Similar to ours, Agrawal et al. (2022) use a few-shot prompt with GPT-3 for the extraction of clinical relations.

2.2 Dual-System Model

There is increasing interest in combining neural and symbolic systems (Marcus, 2018; Lamb et al., 2020; Sarker et al., 2021). Such dual-system models achieved new state-of-the-art results in visual question answering (Goldman et al., 2018; Sampat and Lee, 2018; Yi et al., 2019; Chen et al., 2020b; Ding et al., 2021). In the case of textual problems, to improve LLMs to generate more consistent and coherent sentences, Nye et al. (2021) suggest that generation be decomposed into two parts: candidate sentence generation by an LLM (system 1 thinking) and a logical pruning process (system 2 thinking) implemented via a separate symbolic module. They demonstrate that this neuro-symbolic, dual-process model requires less training data, achieves higher accuracy, and exhibits better generalization. However, the main limitation of their work is that the symbolic module is manually constructed in Python code for the specific task at hand, requiring subtantial efforts. Additionally, their Python symbolic module is not readily reusable or composable. Furthermore, their main results primarily focus on the problem of consistent text generation, rather than evaluating the method on the datasets and comparing it with existing models. This is because writing the world models in Python is not a scalable approach.

In our work, we follow the idea presented in (Nye et al., 2021) but adopt logic programming in place of the System 2 process. We argue that this combination is much more appealing than the approach in (Nye et al., 2021), as it can achieve the promised results without the limitations mentioned above.

2.3 Answer Set Programming

Answer Set Programming (ASP) (Lifschitz, 2008; Brewka et al., 2011) is a declarative logic programming paradigm that has been shown to be effective in knowledge-intensive applications. It is based on the stable model (a.k.a. answer set) semantics of logic programs (Gelfond and Lifschitz, 1988), which could express causal reasoning, default reasoning, aggregates, and various other constraints. There are several efficient solvers, such as CLINGO, DLV, and WASP. We use CLINGO v5.6.0 as the answer set solver. For the language of CLINGO, we refer the reader to the textbook (Lifschitz, 2019) or the CLINGO manual.[1]

It is also known that classical logic-based action formalisms, such as the situation calculus (McCarthy and Hayes, 1969; Reiter, 2001) and the event calculus (Shanahan, 1995), can be formulated as answer set programs. For example, the following is one of the axioms in Discrete Event Calculus stating the commonsense law of inertia, saying that fluent F holds at the next time if there is no action affecting it.

```
% (DEC5)
holds_at(F,T+1) :- timepoint(T), fluent(F),
    holds_at(F,T), -released_at(F,T+1),
    not terminated(F,T).
```

Such a rule is universal and applies to almost all objects.

Answer set programs are also known to be *elaboration tolerant* (McCarthy, 1998). There has been work on modularizing knowledge bases in ASP, such as module theorem (Oikarinen and Janhunen, 2006; Babb and Lee, 2012) and knowledge modules (Baral et al., 2006). While ASP has been widely applied to many reasoning problems, it has not been considered as much in reasoning with natural language text because its input is expected to be strictly in a logical form, giving little flexibility in accepting diverse forms of natural language input.

3 Our Method

We refer to our framework as [LLM]+ASP where [LLM] denotes a large pre-trained network such as GPT-3, which we use as a semantic parser to generate input to the ASP reasoner. Specifically, we assume data instances of the form $\langle S, q, a\rangle$, where S is a context story in natural language, q is a natural language query associated with S, and a is the answer. We use an LLM to convert a problem description (that is, context S and query q) into atomic facts, which are then fed into the ASP solver along with background knowledge encoded as ASP rules. The output of the ASP solver is interpreted as the prediction for the given data instance. Figure 1 illustrates the inference flow in the context of StepGame. The pipeline is simple but general enough to bke applied to various tasks without the need for retraining. It only requires replacing the few-shot prompts to the LLM and the ASP background knowledge with those suitable for the new tasks.

By combining LLMs and ASP in this manner, we enable robust symbolic reasoning that can handle diverse and unprocessed textual input. The ASP knowledge modules remain unaffected by the diverse forms of input text that express the same facts. Our method does not rely on training datasets. Instead, a few examples that turn natural language sentences into atomic facts are sufficient to build a semantic parser due to the learned representations in LLMs. Furthermore, ASP knowledge modules can be reused for different tasks.

3.1 Prompts for Fact Extraction

We use GPT-3 to extract atomic facts from the story and query. Most of the time, giving several examples yields accurate semantic parsing. The following is an example prompt for bAbI.

```
Please parse the following statements into facts
    . The available keywords are: pickup, drop,
    and go.
Sentence: Max journeyed to the bathroom.
Semantic parse: go(Max, bathroom).

Sentence: Mary grabbed the football there.
Semantic parse: pickup(Mary, football).
...
```

We find that GPT-3 is highly tolerable to linguistic variability. For example, in StepGame, GPT-3 can turn various sentences below into the same atomic fact `top_right("C","D")`.

```
C is to the top right of D.
C is to the right and above D at an angle of
    about 45 degrees.
C is at a 45 degree angle to D, in the upper
    righthand corner.
C is directly north east of D.
C is above D at 2 o'clock.
```

In the experiments to follow, we find that the following strategy works well for fact extraction.

1. In general, we find that if the information in a story (or query) can be extracted independently, parsing each sentence separately (using the same prompt multiple times) typically works better than parsing the whole story.

2. There is certain commonsense knowledge that GPT-3 is not able to leverage from the examples in the prompt. In this case, detailing the missing knowledge in the prompt could work. For example, in StepGame, clock numbers are used to denote cardinal directions, but GPT-3 couldn't translate correctly even with a few

[1] https://github.com/potassco/guide/releases.

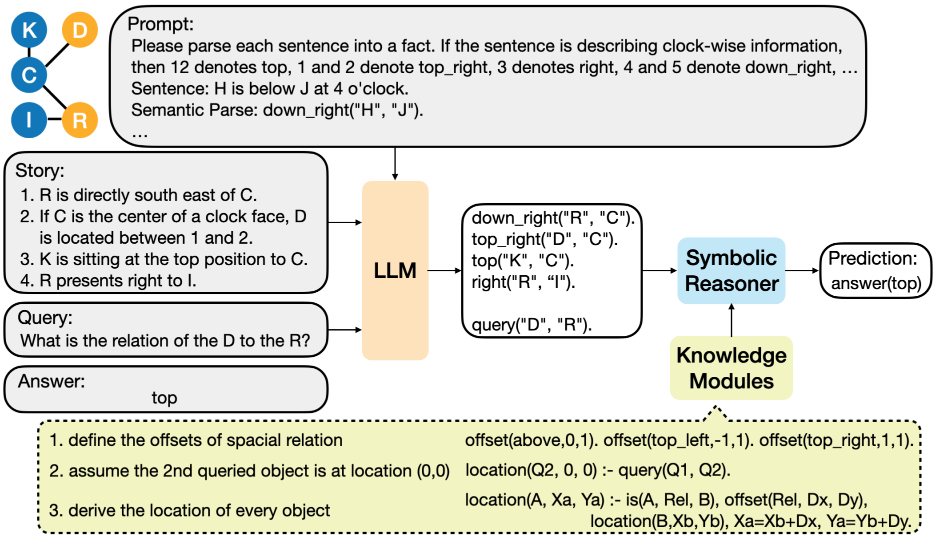

K
D
C
I
R
Prompt:
Please parse each sentence into a fact. If the sentence is describing clock-wise information, then 12 denotes top, 1 and 2 denote top_right, 3 denotes right, 4 and 5 denote down_right, …
Sentence: H is below J at 4 o'clock.
Semantic Parse: down_right("H", "J").
…
Story:
1. R is directly south east of C.
2. If C is the center of a clock face, D is located between 1 and 2.
3. K is sitting at the top position to C.
4. R presents right to I.
Query:
What is the relation of the D to the R?
Answer:
top
LLM
down_right("R", "C").
top_right("D", "C").
top("K", "C").
right("R", "I").
query("D", "R").
Symbolic Reasoner
Prediction:
answer(top)
Knowledge Modules
1. define the offsets of spacial relation
offset(above,0,1). offset(top_left,-1,1). offset(top_right,1,1).
2. assume the 2nd queried object is at location (0,0)
location(Q2, 0, 0) :- query(Q1, Q2).
3. derive the location of every object
location(A, Xa, Ya) :- is(A, Rel, B), offset(Rel, Dx, Dy), location(B,Xb,Yb), Xa=Xb+Dx, Ya=Yb+Dy.

Figure 1: The GPT-3+ASP pipeline for the StepGame dataset.

examples in the prompt. It works after enumerating all cases ("12 denotes top, 1 and 2 denote top_right, 3 denotes right, . . .") in the prompt.

3. Semantic parsing tends to work better if we instruct GPT-3 to use a predicate name that better reflects the intended meaning of the sentence. For example, "A is there and B is at the 5 position of a clock face" is better to be turned into `down_right(B,A)` than `top_left(A,B)` although, logically speaking, the relations are symmetric.

The complete set of prompts for semantic parsing is given in Appendix C.

3.2 Knowledge Modules

Instead of constructing a minimal world model for each task in Python code (Nye et al., 2021), we use ASP knowledge modules. While some knowledge could be lengthy to be described in English, it could be concisely expressed in ASP. For example, the **location** module contains rules for spatial reasoning in a 2D grid space and is used for bAbI, StepGame, and gSCAN. Below is the main rule in the **location** module that computes the location `(Xa,Ya)` of object `A` from the location `(Xb,Yb)` of object `B` by adding the offsets `(Dx,Dy)` defined by the spatial relation `R` between `A` and `B`.

```
location(A, Xa, Ya) :- location(B, Xb, Yb),
    is(A, R, B), offset(R, Dx, Dy),
    Xa=Xb+Dx, Ya=Yb+Dy.
```

The **location** module also includes 9 predefined offsets, e.g., `offset(left,-1,0)`, that can be used to model multi-hop spatial relations of objects or effects of a robot's moving in a 2D space. For example, queries in StepGame are about the spatial relation `R` of object `A` to `B`. Using the **location** module, one can fix `B`'s location to be `(0,0)` and compute the spatial relation `R` based on the location of `A` as follows.

```
location(B, 0, 0) :- query(A, B).
answer(R) :- query(A, B), location(A, X, Y),
    offset(R, Dx, Dy),
    Dx=-1: X<0; Dx=0: X=0; Dx=1: X>0;
    Dy=-1: Y<0; Dy=0: Y=0; Dy=1: Y>0.
```

The second rule above contains six *conditional literals* among which `Dx=-1:X<0` says that "`Dx` must be -1 if `X<0`." For example, if `A`'s location `(X,Y)` is `(-3,0)`, then `(Dx,Dy)` is `(-1,0)` and the answer `R` is `left`. Similar rules can also be applied to bAbI task 17, which asks if `A` is `R` of `B`.

In the above rules, the relation `R` in, e.g., `is(A,R,B)`, is a variable and can be substituted by any binary relation. Such high-order representation turns out to be quite general and applicable to many tasks that query relation or its arguments.

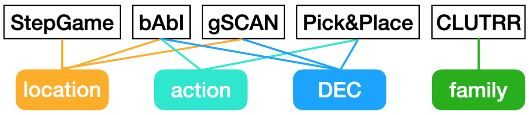


Figure 2: The knowledge modules at the bottom are used in each task on the top.

Figure 2 shows the knowledge modules used in this paper, where **DEC** denotes the Discrete Event Calculus axioms from (Mueller, 2006; Lee and Palla, 2012). In this section, we explained the main rules in the **location** module. The complete ASP knowledge modules are given in Appendix E.

4 Experiments

We apply the method in the previous section to four datasets.[2] As a reminder, our approach involves few-shot in-context learning and does not require training. We use the same pipeline as shown in Figure 1, but with different prompts and knowledge modules for each dataset. For more detailed information about the experimental settings, please refer to the appendix.

4.1 bAbI

The bAbI dataset (Weston et al., 2016) is a collection of 20 QA tasks that have been widely applied to test various natural language reasoning problems, such as deduction, path-finding, spatial reasoning, and counting. State-of-the-art models, such as self-attentive associative-based two-memory model (STM) (Le et al., 2020) and Query-Reduction networks (QRN) (Seo et al., 2017) achieve close to 100% accuracy after training with 10k instances while QRN's accuracy drops to 90% with 1k training instances.

We first designed two GPT-3 baselines, one with few shot prompts (containing a few example questions and answers) and the other with Chain-of-Thought (CoT) prompts (Wei et al., 2022), which state the relevant information to derive the answer.

We also apply GPT-3+ASP. For example, we use GPT-3 to turn "the kitchen is south of the bathroom" into an atomic fact `is(kitchen, southOf, bathroom)` by giving a few examples of the same kind. Regarding knowledge modules, Tasks 1–3, 6–9, 10–14, and 19 are about events over time and use the **DEC** knowledge module. Tasks 4, 17, and 19 require various domain knowledge modules such as **location** and **action** knowledge modules. The remaining tasks do not require domain knowledge and rely only on simple rules to extract answers from parsed facts.

Table 1 compares our method with the two GPT-3 baselines, as well as two state-of-the-art methods on bAbI datasets, STM and QRN. Interestingly, the new GPT-3, text-davinci-003 (denoted GPT-3 (d3)), with basic few-shot prompting achieves 80.34% accuracy, while CoT improves it to 86.18%. GPT-3(d3)+ASP achieves state-of-the-art performance on bAbI with 99.99% average performance among all tasks, producing only two answers that disagree with the labels in the dataset. It turns out that the two questions are malformed since the answers are ambiguous, and our model's answers can be considered correct.[3]

4.2 StepGame

Although bAbI has been extensively tested, it has several problems. Shi et al. (2022) note data leakage between the train and the test sets where named entities are fixed and only a small number of relations are used. Palm et al. (2018) point out that models do not need multi-hop reasoning to solve the bAbI dataset. To address the issues, Shi et al. (2022) propose the StepGame dataset. It is a contextual QA dataset in which the system is required to interpret a story S about spatial relationships among several entities and answers a query q about the relative position of two of those entities, as illustrated in Figure 1. Unlike the bAbI dataset, StepGame uses a large number of named entities, and requires multi-hop reasoning up to as many as 10 reasoning steps.

In the basic form of the StepGame dataset, each story consists of k sentences that describe k spatial relationships between $k+1$ entities in a chain-like shape. In this paper, we evaluate the StepGame dataset with noise, where the original chain is extended with noise statements by branching out with new entities and relations.

Similarly to bAbI, we designed two GPT-3 baselines and applied our method to the StepGame data set. More details on the prompts are available in Appendix C.2.

For each $k \in \{1, \dots, 10\}$, the StepGame dataset with noise consists of 30,000 training samples, 1000 validation samples, and 10,000 test samples. To save the API cost for GPT-3, we only evaluated the two GPT-3 baselines on the first 100 test samples and evaluated our method on the first 1,000 test samples for each $k \in \{1, \dots, 10\}$. Table 2 compares the accuracy of our method with the two baselines of GPT-3 and the current methods, i.e. RN (Santoro et al., 2017), RRN (Palm et al., 2018), UT (Dehghani et al., 2018), STM (Le et al., 2020),

[2] Due to space restriction, we put the experiments about Pick&Place in Appendix A.

[3] See Appendix F.1 for the examples.

Task	GPT-3(d3)	GPT-3(d3)	GPT-3(d3)	STM(Le et al., 2020)	QRN(Seo et al., 2017)	
	Few-Shot	CoT	+ASP	(10k train)	(10k train)	(1k train)
1: Single supporting fact	98.4	97.3	100.0	100.0 ± 0.0	100.0	100.0
2: Two supporting facts	60.8	72.2	100.0	99.79 ± 0.23	100.0	99.3
3: Three supporting facts	39.6	54.1	100.0	97.87 ± 1.14	100.0	94.3
4: Two arg relations	60.4	72.7	100.0	100.0 ± 0.0	100.0	100.0
5: Three arg relations	88.2	89.1	99.8	99.43 ± 0.18	100.0	98.9
6: Yes/no questions	97.4	97.3	100.0	100.0 ± 0.0	100.0	99.1
7: Counting	90.6	88.6	100.0	99.19 ± 0.27	100.0	90.4
8: Lists/sets	96.2	97.1	100.0	99.88 ± 0.07	99.6	94.4
9 : Simple negation	98.4	98.2	100.0	100.0 ± 0.0	100.0	100.0
10: Indefinite knowledge	93.6	92.4	100.0	99.97 ± 0.06	100.0	100.0
11: Basic coreference	93.6	99.2	100.0	99.99 ± 0.03	100.0	100.0
12: Conjunction	88.6	88.8	100.0	99.96 ± 0.05	100.0	100.0
13: Compound coreference	98.4	97.3	100.0	99.99 ± 0.03	100.0	100.0
14: Time reasoning	78.0	91.5	100.0	99.84 ± 0.17	99.9	99.2
15: Basic deduction	57.0	95.0	100.0	100.0 ± 0.0	100.0	100.0
16: Basic induction	90.8	97.5	100.0	99.71 ± 0.15	100.0	47.0
17: Positional reasoning	66.0	70.8	100.0	98.82 ± 1.07	95.9	65.6
18: Size reasoning	89.8	97.1	100.0	99.73 ± 0.28	99.3	92.1
19: Path finding	21.0	28.7	100.0	97.94 ± 2.79	99.9	21.3
20: Agents motivations	100.0	100.0	100.0	100.0 ± 0.0	100.0	99.8
Average	80.34	86.18	**99.99**	99.85	99.70	90.1

Table 1: Test accuracy on 20 tasks in bAbI data

Method	k=1	k=2	k=3	k=4	k=5
RN	22.6	17.1	15.1	12.8	11.5
RRN	24.1	20.0	16.0	13.2	12.3
UT	45.1	28.4	17.4	14.1	13.5
STM	53.4	36.0	23.0	18.5	15.1
TPR-RNN	70.3	46.0	36.1	26.8	24.8
TP-MANN	85.8	60.3	50.2	37.5	31.3
SynSup	**98.6**	**95.0**	**92.0**	79.1	70.3
Few-Shot (d3)	55.0	37.0	25.0	30.0	32.0
CoT (d3)	61.0	45.0	30.0	35.0	35.0
GPT-3(c1)+ASP	44.7	38.8	40.5	58.8	62.4
GPT-3(d2)+ASP	92.6	89.9	89.1	**93.8**	**92.9**

Method	k=6	k=7	k=8	k=9	k=10
RN	11.1	11.5	11.2	11.1	11.3
RRN	11.6	11.4	11.8	11.2	11.7
UT	12.7	12.1	11.4	11.4	11.7
STM	13.8	12.6	11.5	11.3	11.8
TPR-RNN	22.3	19.9	15.5	13.0	12.7
TP-MANN	28.5	26.5	23.7	22.5	21.5
SynSup	63.4	58.7	52.1	48.4	45.7
Few-Shot (d3)	29.0	21.0	22.0	34.0	31.0
CoT (d3)	27.0	22.0	24.0	23.0	25.0
GPT-3(c1)+ASP	57.4	56.2	58.0	56.5	54.1
GPT-3(d2)+ASP	**91.6**	**91.2**	**90.4**	**89.0**	**88.3**

Table 2: Test accuracy on the StepGame test dataset, where (c1), (d2), and (d3) denote text-curie-001, text-davinci-002, and text-davinci-003 models, respectively

TPR-RNN (Schlag and Schmidhuber, 2018), TP-MANN (Shi et al., 2022), and SynSup (with pre-training on the SPARTUN dataset) (Mirzaee and Kordjamshidi, 2022). Surprisingly, the GPT-3 baselines could achieve accuracy comparable to other models (except for SynSup) for large k values. CoT does not always help and decreases the accuracy with big ks. This may be because there is a higher chance of making a mistake in a long chain of thought. GPT-3(d2)+ASP outperforms all state-of-the-art methods and the GPT-3 baselines by a large margin for $k = 4, \dots, 10$. Although SynSup achieves a higher accuracy for $k = 1, 2, 3$, this is misleading due to errors in the dataset. As we analyze below, about 10.7% labels in the data are wrong. The SynSup training makes the model learn to make the same mistakes over the test dataset, which is why its performance looks better than ours.

The modular design of GPT-3+ASP enables us to analyze the reasons behind its wrong predictions. We collected the first 100 data instances for each $k \in \{1, \dots, 10\}$ and manually analyzed the predictions on them. Among 1000 predictions of GPT-3(d2)+ASP, 108 of them disagree with the dataset labels, and we found that 107 of those have errors in the labels. For example, given the story and question "*J and Y are horizontal and J is to the right of Y. What is the relation of the agent Y with the agent J?*", the label in the dataset is "right" while the correct relation should be "left".[4] Recall

[4] The remaining disagreeing case is due to text-davinci-002's mistake. For the sentence, "*if E is the center of a clock face, H is located between 2 and 3.*" text-davinci-002 turns it into "right(H, E)" whereas text-davinci-003 turns it into "top-right(H, E)" correctly. To save API cost for GPT-3, we

that our method is interpretable, so we could easily identify the source of errors.

4.3 CLUTRR

CLUTRR (Sinha et al., 2019) is a contextual QA dataset that requires inferring family relationships from a story. Sentences in CLUTRR are generated using 6k template narratives written by Amazon Mechanical Turk crowd-workers, and thus are more realistic and complex compared to those in bAbI and StepGame.

CLUTRR consists of two subtasks, *systematic generalization* that evaluates stories containing unseen combinations of logical rules (Minervini et al., 2020; Bergen et al., 2021) and *robust reasoning* that evaluates stories with noisy descriptions (Tian et al., 2021). Since we use ASP for logical reasoning, which easily works for any combination of logical rules, we focus on the robust reasoning task.

Method	CLU.	clean	supp.	irre.	disc.
RN	1.0	49	68	50	45
MAC	1.0	63	65	56	40
Bi-att	1.0	58	67	51	57
GSM	1.0	**68.5**	48.6	62.9	52.8
GPT-3(d3)+ASP	1.0	**68.5**	**82.8**	**74.8**	**67.4**
GPT-3(d3)+ASP	1.3	97.0	84.0	92.0	90.0

Table 3: Test accuracy on 4 categories in CLUTRR 1.0 and CLUTRR 1.3 datasets

Table 3 compares our method with RN (Santoro et al., 2017), MAC (Hudson and Manning, 2018), BiLSTM-attention (Sinha et al., 2019), and GSM (Tian et al., 2021) on the original CLUTRR dataset, namely CLUTRR 1.0, in four categories of data instances: clean, supporting, irrelevant, and disconnected (Sinha et al., 2019). Except for our method, all other models are trained on the corresponding category of CLUTRR training data. Although our method achieves similar or higher accuracies in all categories, they are still much lower than we expected.

We found that such low accuracy is due to the clear errors in CLUTRR, originating mostly from errors in the template narratives or the generated family graphs that violate common sense. The authors of CLUTRR recently published CLUTRR 1.3 codes to partially resolve this issue. [5] With the new code, we created a new dataset, namely CLUTRR 1.3, consisting of 400 data instances with 100 for each of the four categories. The last row in Table 3 shows that our method actually performs well on realistic sentences in CLUTRR. Indeed, with our method (using text-davinci-003) on CLUTRR 1.3 dataset, 363 out of 400 predictions are correct, 16 are still wrong due to data mistakes (e.g., the label says "Maryann has an uncle Bruno" while the noise sentence added to the story is "Maryann told her son Bruno to give the dog a bath"), and 21 are wrong due to GPT-3's parsing mistakes (e.g., GPT-3 turned the sentence "Watt and Celestine asked their mother, if they could go play in the pool" into `mother("Watt", "Celestine")`. Since the sentences in CLUTRR 1.3 are more realistic than those in bAbI and StepGame, GPT-3 makes more mistakes even after reasonable efforts of prompt engineering. More details on data errors and GPT-3 errors are available in Appendix F.2 and Appendix D.

Method	clean	supp.	irre.	disc.
DeepProbLog	100	100	100	94
GPT-3(d2)+ASP	100	100	97	97
GPT-3(d3)+ASP	100	100	100	100

Table 4: Test accuracy on CLUTRR-S dataset

We also evaluated our method on a simpler and cleaner variant of the CLUTRR data set, namely CLUTRR-S, that was used as a benchmark problem for a state-of-the-art neuro-symbolic approach DeepProbLog (Manhaeve et al., 2021). Table 4 compares the accuracy of our method and DeepProbLog in all 4 categories of test data. GPT-3(d3)+ASP achieves 100% accuracy, outperforming DeepProbLog without the need for training.

Remark: Due to the modular structure, our method could serve as a data set validation tool to detect errors in a dataset. We detected 107 wrong data instances in the first 1000 data in StepGame and 16 wrong data instances in the 400 data in CLUTRR 1.3.

4.4 gSCAN

The gSCAN dataset (Ruis et al., 2020) poses a task in which an agent must execute action sequences to achieve a goal (specified by a command in a natural language sentence) in a grid-based visual navigation environment. The dataset consists of two tasks, and we evaluate our method on the data splits from the compositional generalization task. There is one shared training set, one test set (split

did not re-run the whole experiments with text-davinci-003.

[5] `https://github.com/facebookresearch/clutrr/tree/develop`

A) randomly sampled from the same distribution of the training set, and seven test sets (splits B to H) with only held-out data instances (i.e., not appearing in the training set) in different ways.

In the gSCAN dataset, each data instance is a tuple $\langle G, q, a \rangle$ where G is the grid configuration (in JSON format) describing the size of the gird, the location and direction of the agent, and the location and features of each object in the grid; q is a query (e.g., “pull a yellow small cylinder hesitantly”); and a is the answer in the form of a sequence of actions (e.g., “turn right, walk, stay, pull, stay, pull, stay”). For each data instance, we (i) use a Python script to extract atomic facts (e.g., `pos(agent,(2,3))`) from the grid configuration G; (ii) extract atomic facts from query q into atomic facts (e.g., `query(pull)`, `queryDesc(yellow)`, `while(hesitantly)`) using GPT-3; and (iii) predict the sequence of actions for this query using ASP. The details of the prompts are given in Appendix C.4.

Method	A	B	C	D	E	F	G	H
GECA	87.60	34.92	78.77	0.00	33.19	85.99	0.00	11.83
DualSys	74.7	81.3	78.1	0.01	53.6	76.2	0.0	21.8
Vilbert+CMA	99.95	99.90	99.25	0.00	99.02	99.98	0.00	22.16
GPT-3(c1)+ASP	98.30	**100**	**100**	**100**	**100**	**100**	**100**	**100**
GPT-3(d2)+ASP	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**

Table 5: Test accuracy on the gSCAN dataset

Table 5 compares the accuracy of our method and the state-of-the-art methods, i.e., GECA (Ruis et al., 2020), DualSys (Nye et al., 2021) and Vilbert+CMA (Qiu et al., 2021), on the gSCAN test dataset in eight splits. To save API cost for GPT-3, we only evaluated the first 1000 data instances of each split. With text-davinci-002, our method GPT-3+ASP achieves 100% accuracy. With text-curie-001, the accuracy is slightly lower, making 17 errors in split A. The errors are of two kinds. The language model fails to extract adverbs in the correct format for 11 data instances (e.g., GPT-3 responded `queryDesc(while spinning)` instead of `while(spinning)`) and didn’t ground the last word in a query for 6 data instances (e.g., for query `walk to a small square`, GPT-3 missed an atomic fact `queryDesc(square)`). Once the parsed results are correct, ASP does not make a mistake in producing plans.

4.5 Findings

The following summarizes the findings of the experimental evaluation.

- Our experiments confirm that LLMs like GPT-3 are still not good at multi-step reasoning despite various prompts we tried. Chain-of-Thought is less likely to improve accuracy when a long chain of thought is required.

- On the other hand, LLMs are surprisingly good at turning a variety of expressions into a "canonical form" of information extraction. This in turn allows ASP knowledge modules to be isolated from linguistic variability in the input.

- Even for generating simple atomic facts, larger models tend to perform better. For example, in StepGame and gSCAN, text-curie-001 performs significantly worse compared to text-davinci-002 (Tables 2 and 5).

- The total amount of knowledge that needs to be encoded for all of the above datasets is not too large. This is in part due to the fact that GPT-3 "normalized" various forms of input sentences for ASP to process and that knowledge modules could be reused across different datasets.

- The modular design of our approach makes it possible to locate the root cause of each failed prediction in the training data and improve upon it. There are three sources of errors: semantic parsing in LLMs, symbolic constraints, and the dataset itself, and we can resolve the first two issues by improving the prompts and updating the constraints, respectively.

- Our framework could serve as a few-shot dataset justifier and corrector. Among all predictions by our method that do not align with the labels, almost all of them (with only a few exceptions discussed in the paper) are due to errors in the dataset.

5 Conclusion

Symbolic logic programming was previously considered limited in its ability to reason from text due to its inability to handle various and ambiguous linguistic expressions. However, combining it with a large language model that has learned distributed representations helps alleviate this problem. The method not only achieves high accuracy but also produces interpretable results, as the source of the errors can be identified. It is also general; by using pre-trained networks with few-shot prompts and reusable knowledge modules, adapting to a new domain does not require extensive training.

The knowledge modules used in our experiments are reusable. For the above experiments, the modules are relatively simple to write, as are the prompts for parsing natural language for LLMs. However, acquiring this kind of knowledge on a massive scale is also an important line of research (Liu and Singh, 2004; Bosselut et al., 2019; Hwang et al., 2021) that needs to be combined. In addition, it is possible to use LLM's code generation capability (Chen et al., 2021) to generate logic program rules, which we leave for future work.

One may think that the logic rules are too rigid. However, there are many weighted or probabilistic rules that can be defeated (Richardson and Domingos, 2006; Fierens et al., 2013; Lee and Wang, 2018). They could be used for more realistic settings, but for the benchmark problems above, they were not needed.

Ethical Considerations

All datasets used in this paper are publicly available. For CLUTRR dataset, the gender information is essential to tell if, e.g., A is B's uncle or niece. We used GPT-3 to predict the genders of persons in each story. Since each story is systematically generated using sampled common first names and sampled sentence templates, it does not reveal any identity. As mentioned, the original CLUTRR dataset had some errors, and we describe carefully the codes and settings of the generated CLUTRR 1.3 dataset in Appendix B.1.

Limitations

The current work requires that knowledge modules be written by hand. Commonly used axioms, such as general knowledge like the commonsense law of inertia expressed by event calculus, can be reused easily, but there are vast amounts of other commonsense knowledge that are not easy to obtain. LLMs could be used to supply this information, but we have not tried. Knowledge graphs, such as ConceptNet (Liu and Singh, 2004), COMET (Bosselut et al., 2019) and ATOMIC (Hwang et al., 2021), can be utilized to populate ASP rules. Like code models, we expect that LLMs could generate ASP code, which we leave for future work.

Also, when using large language models, despite various efforts, sometimes it is not understandable why they do not behave as expected.

Acknowledgements

This work was partially supported by the National Science Foundation under Grant IIS-2006747.

References

Monica Agrawal, Stefan Hegselmann, Hunter Lang, Yoon Kim, and David Sontag. 2022. Large language models are few-shot clinical information extractors. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, page 1998–2022. Association for Computational Linguistics.

Michael Ahn, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2022. Do as I can, not as I say: Grounding language in robotic affordances. In *6th Annual Conference on Robot Learning*.

Joseph Babb and Joohyung Lee. 2012. Module theorem for the general theory of stable models. *Theory and Practice of Logic Programming*, 12(4-5):719–735.

Chitta Baral, Juraj Dzifcak, and Hiro Takahashi. 2006. Macros, macro calls and use of ensembles in modular answer set programming. In *International Conference on Logic Programming*, pages 376–390. Springer.

Leon Bergen, Timothy O'Donnell, and Dzmitry Bahdanau. 2021. Systematic generalization with edge transformers. *Advances in Neural Information Processing Systems*, 34:1390–1402.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. 2019. Comet: Commonsense transformers for knowledge graph construction. In *Association for Computational Linguistics (ACL)*.

Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski. 2011. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Xiang Chen, Ningyu Zhang, Xin Xie, Shumin Deng, Yunzhi Yao, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. 2022. Knowprompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction. In *Proceedings of the ACM Web Conference 2022*, pages 2778–2788.

Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta. 2020a. Low-resource domain adaptation for compositional task-oriented semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5090–5100.

Zhenfang Chen, Jiayuan Mao, Jiajun Wu, Kwan-Yee Kenneth Wong, Joshua B Tenenbaum, and Chuang Gan. 2020b. Grounding physical concepts of objects and events through dynamic visual reasoning. In *International Conference on Learning Representations*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2018. Universal transformers. In *International Conference on Learning Representations*.

Mingyu Ding, Zhenfang Chen, Tao Du, Ping Luo, Josh Tenenbaum, and Chuang Gan. 2021. Dynamic visual reasoning by learning differentiable physics models from video and language. *Advances in Neural Information Processing Systems*, 34.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *54th Annual Meeting of the Association for Computational Linguistics*, pages 33–43. Association for Computational Linguistics (ACL).

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. 2022. Compositional semantic parsing with large language models. *arXiv preprint arXiv:2209.15003*.

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. 2013. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, pages 1–44.

Michael Gelfond and Vladimir Lifschitz. 1988. The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press.

Omer Goldman, Veronica Latcinnik, Ehud Nave, Amir Globerson, and Jonathan Berant. 2018. Weakly supervised semantic parsing with abstract examples. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1809–1819.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and brian ichter. 2022. Inner monologue: Embodied reasoning through planning with language models. In *6th Annual Conference on Robot Learning*.

Drew A Hudson and Christopher D Manning. 2018. Compositional attention networks for machine reasoning. In *International Conference on Learning Representations*.

Jena D Hwang, Chandra Bhagavatula, Ronan Le Bras, Jeff Da, Keisuke Sakaguchi, Antoine Bosselut, and Yejin Choi. 2021. (comet-) atomic 2020: On symbolic and neural commonsense knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6384–6392.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22.

Tomáš Kočiskỳ, Gábor Melis, Edward Grefenstette, Chris Dyer, Wang Ling, Phil Blunsom, and Karl Moritz Hermann. 2016. Semantic parsing with semi-supervised sequential autoencoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1078–1087.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR.

Luis C Lamb, Artur Garcez, Marco Gori, Marcelo Prates, Pedro Avelar, and Moshe Vardi. 2020. Graph neural networks meet neural-symbolic computing: A survey and perspective. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4877–4884.

Hung Le, Truyen Tran, and Svetha Venkatesh. 2020. Self-attentive associative memory. In *International Conference on Machine Learning*, pages 5682–5691. PMLR.

Joohyung Lee and Ravi Palla. 2012. Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. *Journal of Artificial Inteligence Research (JAIR)*, 43:571–620.

Joohyung Lee and Yi Wang. 2018. Weight learning in a probabilistic extension of answer set programs. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 22–31.

Vladimir Lifschitz. 2008. What is answer set programming? In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1594–1597. MIT Press.

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

Hugo Liu and Push Singh. 2004. Conceptnet—a practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pretrain, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys (CSUR)*.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2021. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298:103504.

Gary Marcus. 2018. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*.

John McCarthy. 1998. Elaboration tolerance. In *Working Papers of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning*.

John McCarthy and Patrick Hayes. 1969. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh.

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 55–61.

Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. 2020. Learning reasoning strategies in end-to-end differentiable proving. In *International Conference on Machine Learning*, pages 6938–6949. PMLR.

Roshanak Mirzaee and Parisa Kordjamshidi. 2022. Transfer learning with synthetic corpora for spatial role labeling and reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, page 6148–6165. Association for Computational Linguistics.

Erik Mueller. 2006. *Commonsense reasoning*. Elsevier.

Maxwell Nye, Michael Tessler, Josh Tenenbaum, and Brenden M Lake. 2021. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. *Advances in Neural Information Processing Systems*, 34:25192–25204.

Emilia Oikarinen and Tomi Janhunen. 2006. Modular equivalence for normal logic programs. In *17th European Conference on Artificial Intelligence(ECAI)*, pages 412–416.

Rasmus Palm, Ulrich Paquet, and Ole Winther. 2018. Recurrent relational networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3368–3378.

Linlu Qiu, Hexiang Hu, Bowen Zhang, Peter Shaw, and Fei Sha. 2021. Systematic generalization on gscan: What is nearly solved and what is next? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2180–2188.

Raymond Reiter. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine Learning*, 62(1-2):107–136.

Sebastian Ruder. 2021. Challenges and Opportunities in NLP Benchmarking. `http://ruder.io/nlp-benchmarking`.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M Lake. 2020. A benchmark for systematic generalization in grounded language understanding. *Advances in neural information processing systems*, 33:19861–19872.

Shailaja Sampat and Joohyung Lee. 2018. A model-based approach to visual reasoning on cnlvr dataset. In *Sixteenth International Conference on Principles of Knowledge Representation and Reasoning*.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. 2017. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976.

Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. 2021. Neuro-symbolic artificial intelligence. *AI Communications*, pages 1–13.

Imanol Schlag and Jürgen Schmidhuber. 2018. Learning to reason with third order tensor products. *Advances in neural information processing systems*, 31.

Nathan Schucher, Siva Reddy, and Harm de Vries. 2022. The power of prompt tuning for low-resource semantic parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 148–156.

Min Joon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Query-reduction networks for question answering. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Murray Shanahan. 1995. A circumscriptive calculus of events. *Artif. Intell.*, 77(2):249–284.

Zhengxiang Shi, Qiang Zhang, and Aldo Lipani. 2022. Stepgame: A new benchmark for robust multi-hop spatial reasoning in texts. *Association for the Advancement of Artificial Intelligence*.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen Jr, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7699–7715.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L Hamilton. 2019. Clutrr: A diagnostic benchmark for inductive reasoning from text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4506–4515.

Jidong Tian, Yitian Li, Wenqing Chen, HE Hao, and Yaohui Jin. 2021. A generative-symbolic model for logical reasoning in nlu. In *Is Neuro-Symbolic SOTA still a myth for Natural Language Inference? The first workshop*.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for LLMs on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Chenguang Wang, Xiao Liu, and Dawn Song. 2022. Ielm: An open information extraction benchmark for pre-trained language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, page 8417–8437. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomás Mikolov. 2016. Towards ai-complete question answering: A set of prerequisite toy tasks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967.

Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B Tenenbaum. 2019. CLEVRER: Collision events for video representation and reasoning. In *ICLR*.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Andy Zeng, Adrian Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, et al. 2022. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*.

Luke S Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 658–666.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Wenxuan Zhou and Muhao Chen. 2022. An improved baseline for sentence-level relation extraction. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 161–168, Online only. Association for Computational Linguistics.

Appendix

Section A presents another experiment with robot planning. Section B discusses more details about how we generated CLUTRR dataset and the experimental result on CLUTRR 1.0. Section C presents GPT-3 prompts for semantic parsing. Section D enumerates the errors with GPT-3 in semantic parsing. Section E presents ASP knowledge modules we used for the experiments. Section F enumerates the errors in the datasets.

For bAbI, the prompts for the baseline few-shot prompting can be found in the directory `bAbI_baseline/example_prompts`, while the prompts for chain-of-thought can be found in `bAbI_baseline/COT_prompts_v3`. For StepGame, the prompts for the baseline few-shot prompting and chain-of-thought can be found in the directory `stepGame/prompts`. The following table records the cost for GPT-3 queries used in GPT-3 baselines and our method, where Eng. denotes the engine of GPT-3, c1, d2, d3 denote text-curie-001, text-davinci-002, and text-davinci-003.

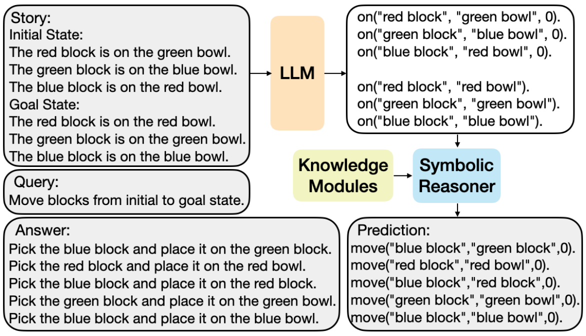
Dataset	Method	Eng.	#Data	Cost
bAbI	Few-Shot	d3	20k	\$190
	CoT	d3	20k	\$280
	GPT-3+ASP	d3	20k	\$41
StepGame	Few-Shot	d3	1k	\$21
	CoT	d3	1k	\$26
	GPT-3+ASP	c1	10k	\$89
	GPT-3+ASP	d2	10k	\$886
CLUTRR 1.0	GPT-3+ASP	d3	879	\$37
CLUTRR 1.3	GPT-3+ASP	d3	400	\$17
CLUTRR-S	GPT-3+ASP	d3	563	\$19
gSCAN	GPT-3+ASP	c1	8k	\$0.2
Pick&Place	Few-Shot	d3	40	\$0.5
	GPT-3+ASP	d3	40	\$0.4

All experiments were conducted on Ubuntu 18.04.2 LTS with two 10-core CPU Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz and four GP104 [GeForce GTX 1080] graphics cards.

All datasets used in this paper are publicly available. The bAbI dataset is under BSD license. The CLUTRR dataset is released under "Attribution-NonCommercial 4.0 International" license. The StepGame dataset doesn't have a specified license. The gSCAN dataset is released under MIT license.

A Robot Planning

Recently, there has been increasing interest in using LLMs to find a sequence of executable actions for robots, aiming to achieve high-level goals expressed in natural language, such as SayCan (Ahn et al., 2022) and Innermonologue (Huang et al., 2022). However, it is worth noting that the actions generated by LLMs tend to be loosely connected and do not take into account the intermediate state changes that occur during the execution of these actions.



Story:
Initial State:
The red block is on the green bowl.
The green block is on the blue bowl.
The blue block is on the red bowl.
Goal State:
The red block is on the red bowl.
The green block is on the green bowl.
The blue block is on the blue bowl.

Query:
Move blocks from initial to goal state.

```
on("red block", "green bowl", 0).
on("green block", "blue bowl", 0).
on("blue block", "red bowl", 0).

on("red block", "red bowl").
on("green block", "green bowl").
on("blue block", "blue bowl").
```

Answer:
Pick the blue block and place it on the green block.
Pick the red block and place it on the red bowl.
Pick the blue block and place it on the red block.
Pick the green block and place it on the green bowl.
Pick the blue block and place it on the blue bowl.

Prediction:
```
move("blue block","green block",0).
move("red block","red bowl",0).
move("blue block","red block",0).
move("green block","green bowl",0).
move("blue block","blue bowl",0).
```

Figure 3: The GPT-3+ASP pipeline for Pick&Place

We based our work on SayCan's open-source virtual tabletop environment[6], where a robot is tasked with achieving a goal, such as "stack the blocks," on a table with colored blocks and bowls. We noticed that the successful plans demonstrated by SayCan are restricted to simple one-step look-ahead plans that do not take into account intermediate state changes.

We randomly sampled 40 data instances of the form $\langle S_i, S_g, L \rangle$ in the Pick&Place domain with 4 to 7 blocks and 3 to 7 bowls, possibly stacked together and with 3 to 10 steps of pick_and_place actions required by the robot to change the initial state S_i to the goal state S_g. Here, the label L is the set of instructions to achieve the goals (e.g., "1. Move the violet block onto the blue block. 2..."). Among 40 data instances, 20 data instances contain only blocks that can be placed on the table while 20 data instances contain both blocks and bowls and assume all blocks must be on the bowls.

The baseline for this dataset follows the method in SayCan's open-source virtual tabletop environment, where GPT-3 is used as the large language model to directly find the sequence of actions from S_i to S_g. However, the baseline fails to find successful plans for all 40 randomly sampled data instances. This result confirms the claim by (Valmeekam et al., 2022) that large language models are not suitable as planners.

We also applied our method to this task. We let GPT-3 turn the states S_i and S_g into atomic facts of the form $on(A, B, 0)$ and $on(A, B)$, respectively.

[6] `https://github.com/google-research/google-research/tree/master/saycan`

Then, an ASP program for the Pick&Place domain is used to find an optimal plan. We found that while GPT-3 has only 0% accuracy in predicting the whole plan, it has 100% accuracy in fact extraction under the provided format. When we apply symbolic reasoning to these extracted atomic facts with an ASP program, we could achieve 100% accuracy on the predicted plans. Details of the prompts are available in Appendix C.5.

Method	Blocks	Blocks+Bowls
GPT-3(d3)	0	0
GPT-3(d3)+ASP	**100**	**100**

Table 6: Test accuracy on the Pick&Place dataset. (d3) denotes the text-davinci-003 model.

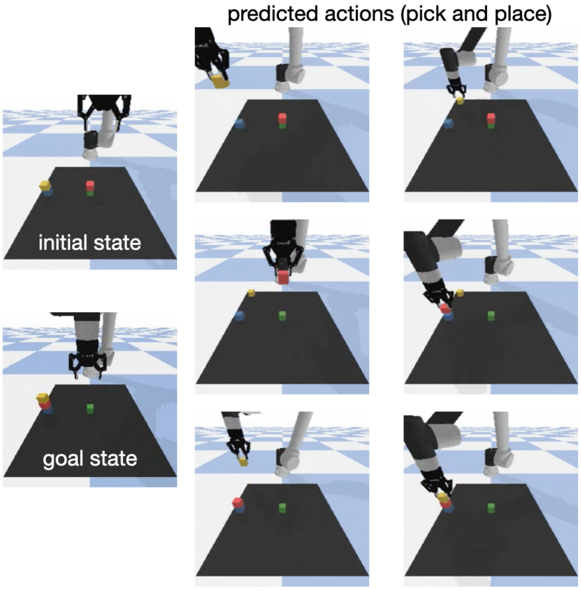


Figure 4: A simple plan predicted by GPT-3+ASP in the Pick&Place domain.

B More about CLUTRR

B.1 CLUTRR 1.3 Data Generation

We used CLUTRR 1.3 codes to generate 400 test data instances. [7] Our generated CLUTRR 1.3 dataset consists of 100 data for each of the four categories: (assuming that the query is asking about the relation between persons A and D)

- clean: each story describes 3 relations in a chain of four persons $A - B - C - D$;
- supporting: each story describes 3 relations in a chain of four persons $A - B - C - D$ as well as an additional relation $X - Y$ such that $X, Y \in \{A, B, C, D\}$ and $X - Y$ is not the queried pair;
- irrelevant: each story describes 3 relations in a chain of four persons $A - B - C - D$ as well as an additional relation $X - Y$ such that $X \in \{A, B, C, D\}$ and $Y \notin \{A, B, C, D\}$;
- disconnected: each story describes 3 relations in a chain of four persons $A - B - C - D$ as well as an additional relation $X - Y$ such that $X, Y \notin \{A, B, C, D\}$.

B.2 Evaluation on CLUTRR 1.0

Training	Testing	BA	GSM	d2	d3
Clean	Clean	58	**69**	63	68
	Supporting	**76**	66	62	62
	Irrelevant	70	**77**	66	71
	Disconnected	49	36	**59**	**59**
Supporting	Supporting	67	49	**83**	**83**
Irrelevant	Irrelevant	51	63	72	**75**
Disconnected	Disconnected	57	53	63	**67**

Table 7: Test accuracy on the CLUTRR dataset. BA denotes BiLSTM-Attention. d2 and d3 denote GPT-3+ASP with text-davinci-002 and text-davinci-003 model.

Table 7 compares the accuracy of our method and the state-of-the-art methods, i.e., BiLSTM-Attention (Sinha et al., 2019) and GSM (with a BiLSTM encoder) (Tian et al., 2021), on the (original) CLUTRR test dataset. Except for our method, all other models are trained on a specific split of the CLUTRR training dataset.

Training	Testing	DP	d2	d3
Clean	Clean	**100**	**100**	**100**
	Supporting	**99**	96	**99**
	Irrelevant	98	99	**100**
	Disconnected	99	98	**100**
Supporting	Supporting	**100**	**100**	**100**
Irrelevant	Irrelevant	**100**	97	**100**
Disconnected	Disconnected	94	97	**100**

Table 8: Test accuracy on the CLUTRR-S dataset. DP denotes DeepProbLog, d2 and d3 denote GPT-3+ASP with the text-davinci-002 and text-davinci-003 model.

Table 8 compares the accuracy of our method and the state-of-the-art method, DeepProbLog (Manhaeve et al., 2021) on the CLUTRR-S test dataset. With GPT-3(d2)+ASP on the CLUTRR-S dataset, 550 out of 563 predictions are correct,

[7] We used the development branch of CLUTRR repository https://github.com/facebookresearch/clutrr/tree/develop.

and 13 are wrong. All errors occur due to the entities in a relation being swapped. For example, we use "`son(A,B)`" to represent "A has a son B" while GPT-3 text-davinci-002 responded with "`son(Robert,Ryan)`" for the sentence "Robert is Ryan's son." On the other hand, text-davinci-003 performed better, with only a single error and 562 out of 563 predictions being correct.

C Prompts for Semantic Parsing

Below, we present the details of the general knowledge of the prompts that we summarized and applied in this paper, followed by some examples.

1. If the information in a story (or query) can be extracted independently, parsing each sentence separately (using the same prompt multiple times) typically works better than parsing the whole story. Since people usually cache all GPT-3 responses to save cost by avoiding duplicated GPT-3 requests for the same prompt, parsing each sentence separately also yields better usage of cached responses. Below are some examples.

    - In most bAbI tasks (except for tasks 11 and 13), the sentences in a story (including the query sentence) are independent of each other. We parse each sentence separately using GPT-3 as in the Appendix C.1.
    - In the stepGame dataset, each sentence in a story describes the spatial relation between 2 objects. There are 4 sentences in a story when $k = 1$ and about 20 sentences when $k = 10$. If we ask GPT-3 to extract all the atomic facts from the whole story, it always misses some atoms or predicts wrong atoms. Since every sentence is independent of each other as shown in Figure 1, we use the following (truncated) prompt multiple times for each data instance where each time `[INPUT]` is replaced with one sentence in the story or the query. This yields a much higher accuracy as in Section 4.3. The complete prompt is available in Appendix C.2.

```
Please parse each sentence into a fact
    . If the sentence is describing
    clock-wise information, then 12
    denotes top, 1 and 2 denote
    top_right, 3 denotes right, ... If
     the sentence is describing
    cardinal directions, then north
    denotes top, ...

Sentence: What is the relation of the
    agent X to the agent K?
Semantic Parse: query("X", "K").

Sentence: H is positioned in the front
     right corner of M.
Semantic Parse: top_right("H", "M").
...
Sentence: [INPUT]
Semantic Parse:
```

However, if some sentences in a story are dependent, splitting them may lead to unexpected results in the GPT-3 response. Below are some examples.

- In bAbI task #11 and #13, a story may contain the two consecutive sentences "Mary went back to the bathroom. After that she went to the bedroom." There is a dependency on the sentences to understand that "she" in the second sentence refers to "Mary" in the first. For this reason, task #11 stories are parsed as a whole. This is similar for task #13.
- In the CLUTRR dataset, a story may contain sentences with coreferences like "Shirley enjoys playing cards with her brother. His name is Henry." where the latter sentence depends on the former one, and a family relation can be correctly extracted only with both sentences. Thus for CLUTRR datasets (i.e., CLUTRR 1.0, CLUTRR 1.3, and CLUTRR-S), we extract the family relations and gender relations from the whole story.

2. There is certain commonsense knowledge that GPT-3 is not aware of, and describing the missing knowledge in the prompt works better than adding examples only. This happens when GPT-3 cannot generalize such knowledge well with a few examples.

    - For example, in StepGame dataset, clock numbers are used to denote cardinal directions, e.g., "H is below J at 4 o'clock" means "H is on the bottom-right of J". Such knowledge in the dataset is not well captured by GPT-3 and enumerating examples in the prompt doesn't work well. On the other hand, describing such knowledge at the beginning of the prompt as shown in Appendix C.2 increases the accuracy by a large margin.

C.1 bAbI

For bAbI dataset, there are two prompts for each task, corresponding to the context and query. Each prompt has a consistent set of basic instructions followed by example pairs of text and parsed text. Below are the prompts used to parse the context and query facts from a story and query, where [Input] at the end is replaced with the story in each test data instance. We only present the prompts for Tasks 1,2, and 3. The rest of the prompts can be found in the repository in https://github.com/azreasoners/LLM-ASP/blob/main/bAbI/GPT_prompts.py.

Tasks 1/2/3 (Context)

```
Please parse the following statements into facts
    . The available keywords are: pickup, drop,
    and go.
Sentence: Max journeyed to the bathroom.
Semantic parse: go(Max, bathroom).

Sentence: Mary grabbed the football there.
Semantic parse: pickup(Mary, football).

Sentence: Bob picked up the apple.
Semantic parse: pickup(Bob, apple).

Sentence: Susan dropped the milk.
Semantic parse: drop(Susan, milk).

Sentence: Bob got the football there.
Semantic parse: pickup(Bob, football).

Sentence: Max left the cup.
Semantic parse: drop(Max, cup).

Sentence: Kevin put down the pie there.
Semantic parse: drop(Kevin, pie).

Sentence: John took the football there.
Semantic parse: pickup(John, football).

Sentence: [INPUT]
Semantic parse:
```

Task 1 (Query)

```
Please parse the following questions into query
    facts. The available keywords are:
    whereAgent.
Sentence: Where is Mary?
Semantic parse: whereAgent(Mary).

Sentence: Where is Daniel?
Semantic parse: whereAgent(Daniel).

Sentence: Where is Sandra?
Semantic parse: whereAgent(Sandra).

Sentence: Where is John?
Semantic parse: whereAgent(John).

Sentence: [INPUT]
Semantic parse:
```

Task 2 (Query)

```
Please parse the following questions into query
    facts. The available keywords are: loc.
Sentence: Where is the toothbrush?
Semantic parse: loc(toothbrush).

Sentence: Where is the milk?
Semantic parse: loc(milk).

Sentence: Where is the apple?
Semantic parse: loc(apple).

Sentence: Where is the football?
Semantic parse: loc(football).

Sentence: [INPUT]
Semantic parse:
```

Task 3 (Query)

```
Please parse the following questions into query
    facts. The available keywords are: loc.
Sentence: Where was the football before the
    bathroom?
Semantic parse: before(football,bathroom).

Sentence: Where was the apple before the garden?
Semantic parse: before(apple,garden).

Sentence: Where was the milk before the kitchen?
Semantic parse: before(milk,kitchen).

Sentence: Where was the apple before the bedroom
    ?
Semantic parse: before(apple,bedroom).

Sentence: Where was the football before the
    hallway?
Semantic parse: before(football,hallway).

Sentence: [INPUT]
Semantic parse:
```

C.2 StepGame

For the StepGame dataset, there is only one prompt below to extract the location relations among objects. All example sentences are from the training data in (the noise split of) the original StepGame dataset.[8] The [Input] at the end of the prompt is replaced with each sentence in a test data instance.

```
Please parse each sentence into a fact. If the
    sentence is describing clock-wise
    information, then 12 denotes top, 1 and 2
    denote top_right, 3 denotes right, 4 and 5
```

[8] https://github.com/ZhengxiangShi/StepGame/tree/main/Code/babi_format/noise

```
    denote down_right, 6 denotes down, 7 and 8
    denote down_left, 9 denote left, 10 and 11
    denote top_left. If the sentence is
    describing cardinal directions, then north
    denotes top, east denotes right, south
    denotes down, and west denotes left. If the
    sentence is a question, the fact starts with
     query. Otherwise, the fact starts with one
    of top, down, left, right, top_left,
    top_right, down_left, and down_right.

Sentence: What is the relation of the agent X to
     the agent K?
Semantic Parse: query("X", "K").

Sentence: H is positioned in the front right
    corner of M.
Semantic Parse: top_right("H", "M").

Sentence: F is on the left side of and below Q.
Semantic Parse: down_left("F", "Q").

Sentence: Y and I are parallel, and Y is on top
    of I.
Semantic Parse: top("Y", "I").

Sentence: V is over there with T above.
Semantic Parse: top("T", "V").

Sentence: V is slightly off center to the top
    left and G is slightly off center to the
    bottom right.
Semantic Parse: top_left("V", "G").

Sentence: The objects S and A are over there.
    The object S is lower and slightly to the
    left of the object A.
Semantic Parse: down_left("S", "A").

Sentence: D is diagonally below Z to the right
    at a 45 degree angle.
Semantic Parse: down_right("D", "Z").

Sentence: V is at A's 9 o'clock.
Semantic Parse: left("V", "A").

Sentence: J is at O's 6 o'clock.
Semantic Parse: down("J", "O").

Sentence: H is below J at 4 o'clock.
Semantic Parse: down_right("H", "J").

Sentence: O is there and C is at the 5 position
    of a clock face.
Semantic Parse: down_right("C", "O").

Sentence: If H is the center of a clock face, B
    is located between 10 and 11.
Semantic Parse: top_left("B", "H").

Sentence: [Input]
Semantic Parse:
```

C.3 CLUTRR

For CLUTRR dataset, there are two prompts to extract the family relations and genders from a story respectively. All example stories in both prompts are from the training data “data_06b8f2a1/2.2,2.3_train.csv” in the original CLUTRR dataset.[9] Below is the prompt to extract family relations from a story where [Input] at the end is replaced with the story in each test data instance.

```
Given a story, extract atomic facts of the form
    relation("Person", "Person"). Example
    relations are: father, mother, parent, son,
    daughter, child, grandfather, grandmother,
    grandson, granddaughter, wife, husband,
    spouse, sibling, nephew, niece, uncle, aunt,
     child_in_law, and parent_in_law.

Story: [Verdie] waved good bye to her dad [Henry
    ] for the day and went next door with her
    sister [Amanda]. [Henry]'s daughter, [Amanda
    ], went to the city this weekend. She spent
    her time there visiting her grandfather, [
    Kyle], and had a wonderful time with him.
Semantic Parse: father("Verdie", "Henry").
    sister("Verdie", "Amanda"). daughter("Henry
    ", "Amanda"). grandfather("Amanda", "Kyle").

Story: [Michelle] was excited for today, its her
     daughter's, [Theresa], spring break. She
    will finally get to see her. [Michael] was
    busy and sent his wife, [Marlene], instead.
    [Kristen] loved to care for her newborn
    child [Ronald]. [Eric]'s son is [Arthur].
Semantic Parse: daughter("Michelle", "Theresa").
     wife("Michael", "Marlene"). child("Kristen
    ", "Ronald"). son("Eric", "Arthur").

Story: [Vernon] was present in the delivery room
     when his daughter [Raquel] was born, but
    when his daughter [Constance] was born he
    was too sick. [Vernon] and his daughter [
    Margaret] went to the movies. [Constance], [
    Margaret]'s sister, had to stay home as she
    was sick.
Semantic Parse: daughter("Vernon", "Raquel").
    daughter("Vernon", "Constance"). daughter("
    Vernon", "Margaret"). sister("Margaret", "
    Constance").

Story: [Eric] who is [Carl]'s father grounded [
    Carl] after finding out what [Carl] had done
     at school. [Ronald] was busy planning a 90
    th birthday party for his aunt, [Theresa]. [
    Eric] and his son [Carl] went to the park
    and saw [Eric]'s father [Kyle] there with
    his dog.
Semantic Parse: father("Carl", "Eric"). aunt("
    Ronald", "Theresa"). son("Eric", "Carl").
    father("Eric", "Kyle").

Story: [Shirley] and [Edward] are siblings and
    best friends. They do everything together. [
    Henry] walked his daughters [Amanda] and [
    Michelle] to school. [Kyle] enjoys watching
    movies with his son's daughter. Her name is
     [Amanda].
Semantic Parse: sibling("Shirley", "Edward").
    daughter("Henry", "Amanda"). daughter("Henry
```

[9] The original CLUTRR data is available in https://github.com/facebookresearch/clutrr.

```
    ", "Michelle"). granddaughter("Kyle", "
    Amanda").

Story: [Raquel] and her brother [Casey] took her
     grandmother [Karen] to the store to buy a
    new dress. [Karen] and her husband [Kyle]
    just celebrated 10 years of marriage. [Karen
    ] loves her grandson, [Casey], and he loves
    her too.
Semantic Parse: brother("Raquel", "Casey").
    grandmother("Raquel", "Karen"). husband("
    Karen", "Kyle"). grandson("Karen", "Casey").

Story: [Allen]'s father, [Eric], bought him some
     ice cream. [Karen] was baking cookies for
    her grandson, [Allen]. [Allen]'s brother [
    Arthur] came home from school, so she baked
    some extra for him, too. [Eric]'s son, [
    Arthur], was ill and needed to be picked up
    at school. [Eric] hurried to his side.
Semantic Parse: father("Allen", "Eric").
    grandson("Karen", "Allen"). brother("Allen",
     "Arthur"). son("Eric", "Arthur").

Story: [Karen] was spending the weekend with her
     grandson, [Eddie]. [Eddie]'s sister [
    Michelle] was supposed to come too, but she
    was busy and could n't make it. [Theresa]
    took her daughter, [Michelle], out to High
    Tea yesterday afternoon. [Eddie]'s mother [
    Theresa] baked brownies for dessert after
    they had dinner.
Semantic Parse: grandson("Karen", "Eddie").
    sister("Eddie", "Michelle"). daughter("
    Theresa", "Michelle"). mother("Eddie", "
    Theresa").

Story: [Input]
Semantic Parse:
```

We also use a variant of the above prompt to extract the gender of each person in a story. The prompt context is a bit simpler as there are only two genders. The examples are the same while the `Semantic Parse` result is simply replaced with the atomic facts about gender information. Below is the prompt to extract the gender of each person in a story where `[Input]` is replaced with the story in each test data instance.

```
Given a story, extract atomic facts of the form
    male("Person") or female("Person") for every
     person that appears in the sentences.

Story: [Verdie] waved good bye to her dad [Henry
    ] for the day and went next door with her
    sister [Amanda]. [Henry]'s daughter, [Amanda
    ], went to the city this weekend. She spent
    her time there visiting her grandfather, [
    Kyle], and had a wonderful time with him.
Semantic Parse: female("Verdie"). male("Henry").
     female("Amanda"). male("Kyle").

Story: [Michelle] was excited for today, its her
     daughter's, [Theresa], spring break. She
    will finally get to see her. [Michael] was
    busy and sent his wife, [Marlene], instead.
    [Kristen] loved to care for her newborn
    child [Ronald]. [Eric]'s son is [Arthur].
Semantic Parse: female("Michelle"). female("
    Theresa"). male("Michael"). female("Marlene
    "). female("Kristen"). male("Ronald"). male
    ("Eric"). male("Arthur").

Story: [Vernon] was present in the delivery room
     when his daughter [Raquel] was born, but
    when his daughter [Constance] was born he
    was too sick. [Vernon] and his daughter [
    Margaret] went to the movies. [Constance], [
    Margaret]'s sister, had to stay home as she
    was sick.
Semantic Parse: male("Vernon"). female("Raquel")
    . female("Constance"). female("Margaret").

Story: [Eric] who is [Carl]'s father grounded [
    Carl] after finding out what [Carl] had done
     at school. [Ronald] was busy planning a 90
    th birthday party for his aunt, [Theresa]. [
    Eric] and his son [Carl] went to the park
    and saw [Eric]'s father [Kyle] there with
    his dog.
Semantic Parse: male("Eric"). male("Carl"). male
    ("Ronald"). female("Theresa"). male("Kyle").

Story: [Shirley] and [Edward] are siblings and
    best friends. They do everything together. [
    Henry] walked his daughters [Amanda] and [
    Michelle] to school. [Kyle] enjoys watching
    movies with his son's daughter. Her name is
    [Amanda].
Semantic Parse: female("Shirley"). male("Edward
    "). male("Henry"). female("Amanda"). female
    ("Michelle"). male("Kyle").

Story: [Raquel] and her brother [Casey] took her
     grandmother [Karen] to the store to buy a
    new dress. [Karen] and her husband [Kyle]
    just celebrated 10 years of marriage. [Karen
    ] loves her grandson, [Casey], and he loves
    her too.
Semantic Parse: female("Raquel"). male("Casey").
     female("Karen"). male("Kyle").

Story: [Allen]'s father, [Eric], bought him some
     ice cream. [Karen] was baking cookies for
    her grandson, [Allen]. [Allen]'s brother [
    Arthur] came home from school, so she baked
    some extra for him, too. [Eric]'s son, [
    Arthur], was ill and needed to be picked up
    at school. [Eric] hurried to his side.
Semantic Parse: male("Allen"). male("Eric").
    female("Karen"). male("Arthur").

Story: [Karen] was spending the weekend with her
     grandson, [Eddie]. [Eddie]'s sister [
    Michelle] was supposed to come too, but she
    was busy and could n't make it. [Theresa]
    took her daughter, [Michelle], out to High
    Tea yesterday afternoon. [Eddie]'s mother [
    Theresa] baked brownies for dessert after
    they had dinner.
Semantic Parse: female("Karen"). male("Eddie").
    female("Michelle"). female("Theresa").

Story: [Input]
Semantic Parse:
```

For CLUTRR-S dataset, i.e., the simpler version of the CLUTRR dataset from DeepProbLog (Manhaeve et al., 2021) repository, there are also two prompts below to extract the family relations and genders from a story respectively.[10] All example stories in both prompts are from the training data "data_a7d9402e/2.2,2.3_train.csv".

```
Given a story, extract atomic facts of the form
    relation("Person", "Person") about family
    relationships that appear in the sentences.

Story: [Mervin] is [Robert]'s father. [Robert]
    is the father of [Jim]. [Jon] is [Robert]'s
     brother. [Mervin] is the father of [Jon].
Semantic Parse: father("Robert", "Mervin").
    father("Jim", "Robert"). brother("Robert", "
    Jon"). father("Jon", "Mervin").

Story: [Brooke] is [Cheryl]'s sister. [Jon] is
    the father of [Brooke]. [Melissa] is [Jon]'
    s mother. [Jon] is [Cheryl]'s father.
Semantic Parse: sister("Cheryl", "Brooke").
    father("Brooke", "Jon"). mother("Jon", "
    Melissa"). father("Cheryl", "Jon").

Story: [Jon] is [Carol]'s brother. [Carol] is [
    Joyce]'s mother. [Helen] is [Carol]'s
    sister. [Helen] is a sister of [Jon].
Semantic Parse: brother("Carol", "Jon"). mother
     ("Joyce", "Carol"). sister("Carol", "Helen")
     . sister("Jon", "Helen").

Story: [Melissa] is [Glenn]'s grandmother. [
    Melissa] is the mother of [Calvin]. [Glenn]
     is a son of [Lila]. [Calvin] is [Glenn]'s
    father.
Semantic Parse: grandmother("Glenn", "Melissa").
     mother("Calvin", "Melissa"). son("Lila", "
    Glenn"). father("Glenn", "Calvin").

Story: [Margaret] has a brother named [William].
      [William] is [Carol]'s son. [Margaret] is
     [Carol]'s daughter. [Lila] is the aunt of
     [William].
Semantic Parse: brother("Margaret", "William").
    son("Carol", "William"). daughter("Carol", "
    Margaret"). aunt("William", "Lila").

Story: [Stephanie] is a sister of [Lois]. [Lois
    ] is [Theresa]'s sister. [Helen] is [Lois]'
    s mother. [Helen] is [Stephanie]'s mother.
Semantic Parse: sister("Lois", "Stephanie").
    sister("Theresa", "Lois"). mother("Lois", "
    Helen"). mother("Stephanie", "Helen").

Story: [Jon] is [Elias]'s brother. [Michael] is
     a son of [Helen]. [Jon] is the uncle of [
    Michael]. [Elias] is the father of [Michael
    ].
Semantic Parse: brother("Elias", "Jon"). son("
    Helen", "Michael"). uncle("Michael", "Jon").
     father("Michael", "Elias").

Story: [Carol] has a son called [William]. [
    Melissa] is the mother of [Jon]. [Jon] is
    the uncle of [William]. [Carol] has a
    brother named [Jon].
Semantic Parse: son("Carol", "William"). mother
     ("Jon", "Melissa"). uncle("William", "Jon").
     brother("Carol", "Jon").

Story: [Robert] is the father of [Jim]. [Robert
    ] has a daughter called [Ashley]. [Elias]
    is [Robert]'s brother. [Elias] is the uncle
     of [Ashley].
Semantic Parse: father("Jim", "Robert").
    daughter("Robert", "Ashley"). brother("
    Robert", "Elias"). uncle("Ashley", "Elias").

Story: [Elias] is the father of [Carlos]. [
    Elias] is the father of [Andrew]. [Andrew]
    is [Carlos]'s brother. [Jon] is a brother
    of [Elias].
Semantic Parse: father("Carlos", "Elias").
    father("Andrew", "Elias"). brother("Carlos",
     "Andrew"). brother("Elias", "Jon").

Story: [Jon] is the father of [Ben]. [James] is
     [Kevin]'s brother. [Ben] is a brother of [
    James]. [Jon] is [James]'s father.
Semantic Parse: father("Ben", "Jon"). brother("
    Kevin", "James"). brother("James", "Ben").
    father("James", "Jon").

Story: [Carol] has a sister named [Lila]. [
    William] is [Carol]'s son. [Helen] is [Lila
    ]'s sister. [Lila] is [William]'s aunt.
Semantic Parse: sister("Carol", "Lila"). son("
    Carol", "William"). sister("Lila", "Helen").
     aunt("William", "Lila").

Story: [Calvin] is [Bruce]'s father. [Elias] is
     [Calvin]'s brother. [Calvin] is [Kira]'s
    father. [Kira] is [Bruce]'s sister.
Semantic Parse: father("Bruce", "Calvin").
    brother("Calvin", "Elias"). father("Kira", "
    Calvin"). sister("Bruce", "Kira").

Story: [Carol] is a sister of [Helen]. [Carol]
    is [Carlos]'s aunt. [Lila] is [Carol]'s
    sister. [Carlos] is [Helen]'s son.
Semantic Parse: sister("Helen", "Carol"). aunt("
    Carlos", "Carol"). sister("Carol", "Lila").
    son("Helen", "Carlos").

Story: [Input]
Semantic Parse:
```

Note that, although the sentences in the CLUTRR-S dataset is much simpler than those in CLUTRR dataset, we don't achieve 100% accuracy in GPT-3 responses with the above long prompt. This is partially because the above prompt violates prompting strategy 3 in Section 3 as the order of names in a binary relation in sentences is mostly following "`relationOf(A,B)`" instead of "`relation(B,A)`".

```
Given a story, extract atomic facts of the form
    male("Person") or female("Person") for every
```

[10] The CLUTRR-S dataset is from https://github.com/ML-KULeuven/deepproblog/tree/master/src/deepproblog/examples/CLUTRR/data.

```
     person that appears in the sentences.

Story: [Jon] is [Carol]'s brother.  [Mervin] has
     a daughter called [Carol].  [Chantell] is a
     daughter of [Jon].  [Mervin] has a son
    called [Jon].
Semantic Parse: male("Jon"). female("Carol").
    male("Mervin"). female("Chantell").

Story: [Melissa] is [Glenn]'s grandmother.  [
    Melissa] is the mother of [Calvin].  [Glenn]
     is a son of [Lila].  [Calvin] is [Glenn]'s
    father.
Semantic Parse: female("Melissa"). male("Glenn")
    . male("Calvin"). female("Lila").

Story: [Input]
Semantic Parse:
```

C.4 gSCAN

For gSCAN dataset, there is only one prompt below to extract the command in each data instance. All example sequences are from the training data.[11] The `[Input]` at the end of the prompt is replaced with the command in each test data instance.

```
Please parse each sequence of words into facts.

Sequence: pull a yellow small circle
Semantic Parse: query(pull). queryDesc(yellow).
    queryDesc(small). queryDesc(circle).

Sequence: push a big square
Semantic Parse: query(push). queryDesc(big).
    queryDesc(square).

Sequence: push a green small square cautiously
Semantic Parse: query(push). queryDesc(green).
    queryDesc(small). queryDesc(square). while(
    cautiously).

Sequence: pull a circle hesitantly
Semantic Parse: query(pull). queryDesc(circle).
    while(hesitantly).

Sequence: walk to a yellow big cylinder while
    spinning
Semantic Parse: query(walk). queryDesc(yellow).
    queryDesc(big). queryDesc(cylinder). while(
    spinning).

Sequence: push a big square while zigzagging
Semantic Parse: query(push). queryDesc(big).
    queryDesc(square). while(zigzagging).

Sequence: push a cylinder hesitantly
Semantic Parse: query(push). queryDesc(cylinder)
    . while(hesitantly).

Sequence: [Input]
Semantic Parse:
```

C.5 Pick&Place

For the Pick&Place dataset, there are two prompts below to extract the atomic facts from the initial state and the goal state, respectively.

```
Turn each sentence into an atomic fact of the
    form on(A, B, 0).

Sentence: The red block is on the yellow bowl.
Semantic Parse: on("red block", "yellow bowl",
    0).

Sentence: The violet block is on the blue block.
Semantic Parse: on("violet block", "blue block",
     0).

Sentence: [INPUT]
Semantic Parse:
```

```
Turn each sentence into an atomic fact of the
    form on(A, B).

Sentence: The red block is on the yellow bowl.
Semantic Parse: on("red block", "yellow bowl").

Sentence: The violet block is on the blue block.
Semantic Parse: on("violet block", "blue block")
    .

Sentence: [INPUT]
Semantic Parse:
```

For each sentence in the initial or goal state, we replace `[INPUT]` in the corresponding prompt above with this sentence and request GPT-3 to extract a single atomic fact. The union of these atomic facts extracted from all sentences is then used in the symbolic reasoner module to find an optimal plan.

For the GPT-3 baseline, we use the following prompt to let GPT-3 directly find a plan where `[INPUT]` at the end of the prompt is replaced with the initial and goal state of the queried data instance.

```
Find a shortest plan to move blocks from an
    initial state to a goal state. Note that you
     cannot move a block if anything is on it.
    You cannot move a block onto a target block
    or bowl if there is anything is on the
    target block or bowl. At most two blocks can
     be placed in the same bowl with one on top
    of the other.

# Initial State:
Nothing is on the green bowl.
The violet block is on the blue bowl.
The blue block is on the violet bowl.
The green block is on the blue block.

# Goal State:
The violet block is on the green bowl.
The green block is on the violet block.
The blue block is on the blue bowl.
```

[11] https://github.com/LauraRuis/groundedSCAN/tree/master/data/compositional_splits.zip

```
Nothing is on the violet bowl.

Plan:
1. Move the violet block onto the green bowl.
2. Move the green block onto the violet block.
3. Move the blue block onto the blue bowl.

# Initial State:
Nothing is on the blue bowl.
The yellow block is on the green bowl.
The green block is on the violet bowl.
The violet block is on the green block.
The blue block is on the yellow bowl.
The red block is on the blue block.

# Goal State:
The yellow block is on the blue bowl.
The green block is on the yellow block.
The red block is on the green bowl.
Nothing is on the violet bowl.
The blue block is on the yellow bowl.
The violet block is on the blue block.

Plan:
1. Move the yellow block onto the blue bowl.
2. Move the red block onto the green bowl.
3. Move the violet block onto the blue block.
4. Move the green block onto the yellow block.

[INPUT]

Plan:
```

D GPT-3 Errors in Semantic Parsing

In this section, we group and record the errors in the GPT-3 responses in tables where each row records a 3-tuple $\langle$ dataset, sentence(s), GPT-3 response $\rangle$. In this section, we list the following.

- all 21 errors for the CLUTRR 1.3 dataset with text-davinci-003;
- the single mistake in the first 100 data instances for every $k \in \{1, \ldots, 10\}$ in the StepGame dataset with text-davinci-002.

D.1 Argument misorder

A common mistake in the GPT-3 response is that the relation and arguments for an atom are correctly extracted, but the order of the arguments is incorrect. Such mistakes can be greatly alleviated by proper few-shot prompting where the orders of arguments in the example target atoms follow their orders in the stories.

CLUTRR 1.3	[Erasmus] bought to dress for his father [Joeseph]	father("Joeseph", "Erasmus")
CLUTRR 1.3	[Baldwin] bought to dress for his father [Sidney]	father("Sidney", "Baldwin")
CLUTRR 1.3	[Elizabeth] is the uncle of [Leopold].	uncle("Elizabeth", "Leopold")

There are only 3 errors in CLUTRR 1.3 due to argument misorder. The first 2 mistakes are indeed due to their missing periods at the end of the sentences – if we simply add the periods back, their GPT-3 responses would become correct.

D.2 Wrong relation

Sometimes the arguments are correct, but the relations extracted by GPT-3 are incorrect or cannot be recognized by the ASP program.

CLUTRR 1.3	[Leila] and [Enoch] have been married for 35 years.	married("Leila", "Enoch")
CLUTRR 1.3	[Lottie] and her grandsons [Clinton] and [Enoch] picked berries from the garden.	grandsons("Lottie", "Clinton") grandsons("Lottie", "Enoch")
CLUTRR 1.3	[Felix] loved going to the store with his mom [Christena].	mom("Felix", "Christena")
CLUTRR 1.3	[Archibald] asked his father, [Abraham], to watch over his two sons, [Justus] and [Merle], for the weekend.	sons("Archibald", "Justus", "Merle")
StepGame	If E is the center of a clock face, H is located between 2 and 3.	right("H", "E")

These kinds of mistake may be resolved by restricting the space of possible relations. For example, the mistakes in the first four rows can be resolved by simply adding the sentence "Use spouse("Person", "Person") if two persons are couples." in the prompt.

D.3 Ambiguious or incorrect co-reference

CLUTRR 1.3	[Katherine] was eating lunch with his son [Daniel] and his wife [Adelle].	son("Katherine", "Daniel") wife("Daniel", "Adelle")
CLUTRR 1.3	[Sylvester] and his son [Rollie] went to his mother [Ottilia]'s home	son("Sylvester", "Rollie") mother("Rollie", "Ottilia")
CLUTRR 1.3	[Elsie] and [Lewis] did the Father daughter dance at the recital and his wife [Dollie] was so happy she cried	father("Lewis", "Dollie") wife("Lewis", "Dollie")

D.4 Anonymous argument

CLUTRR 1.3	[Abe] is proud of his daughter's son. His name is [Johnie].	son("Johnie", "Abe")
CLUTRR 1.3	[Watt] and [Celestine] asked their mother, if they could go play in the pool.	mother("Watt", "Celestine")
CLUTRR 1.3	[Davie] told his wife [Kitty] to pick up their daughter.	wife("Davie", "Kitty") daughter("Davie", "Kitty")
CLUTRR 1.3	[Ethyl] just got married to her now husband, and in the process she got herself a daughter-in-law named [Henry].	husband("Ethyl", "Michael"). daughter_in_law("Ethyl", "Henry")
CLUTRR 1.3	[Barnett] and [Manie] asked their mother, if they could go play in the pool.	mother("Barnett", "Manie")
CLUTRR 1.3	[Santiago] told his wife [Octavia] to pick up their daughter.	wife("Santiago", "Octavia") daughter("Santiago", "Octavia")

D.5 Missed to generate some atoms

CLUTRR 1.3	[Elizabeth] bought to dress for his father [Leopold] [Orson] took his wife [Abbie] out for dinner.	father("Leopold", "Orson")
CLUTRR 1.3	[Asa] felt lonely when his wife [Magdalena] was gone to see her mother [Josiephine].	wife("Asa", "Magdalena")
CLUTRR 1.3	[Warner]'s father, [Johnny], and grandfather, [Bryant], went hiking during the first weekend of spring.	male("Johnny") male("Bryant")
CLUTRR 1.3	[Hollie] and [Rosanna], the happy couple, just got married last week.	–
CLUTRR 1.3	[Violet] took her brother [Travis] to the park, but left her sister [Serena] at home.	brother("Violet", "Travis")

E ASP Knowledge Modules

E.1 Discrete Event Calculus (DEC) Axioms Module

```
% (DEC1)
stopped_in(T1,F,T2) :- timepoint(T),
                       timepoint(T1),
                       timepoint(T2),
                       fluent(F),
                       event(E),
                       happens(E,T),
                       T1<T,
                       T<T2,
                       terminates(E,F,T).

% (DEC2)
started_in(T1,F,T2) :- timepoint(T),
                       timepoint(T1),
                       timepoint(T2),
                       fluent(F),
                       event(E),
                       happens(E,T),
                       T1<T,
                       T<T2,
                       initiates(E,F,T).

% (DEC3)
holds_at(F2,T1+T2) :- timepoint(T1),
                      timepoint(T2),
                      fluent(F1),
                      fluent(F2),
                      event(E),
                      happens(E,T1),
                      initiates(E,F1,T1),
                      0<T2,
                      trajectory(F1,T1,F2,T2),
                      not stopped_in(T1,F1,T1+T2
                          ).

% (DEC4)
holds_at(F2,T1+T2) :- timepoint(T1),
                      timepoint(T2),
                      fluent(F1),
                      fluent(F2),
                      event(E),
                      happens(E,T1),
                      terminates(E,F1,T1),
                      0<T2,
                      anti_trajectory(F1,T1,F2,
                          T2),
                      not started_in(T1,F1,T1+T2
                          ).

initiated(F,T) :- timepoint(T),
                  fluent(F),
                  event(E),
                  happens(E,T),
                  initiates(E,F,T).

terminated(F,T) :- timepoint(T),
                   fluent(F),
                   event(E),
                   happens(E,T),
                   terminates(E,F,T).

released(F,T) :- timepoint(T),
                 fluent(F),
                 event(E),
                 happens(E,T),
                 releases(E,F,T).

% (DEC5)
holds_at(F,T+1) :- timepoint(T),
                   fluent(F),
                   holds_at(F,T),
                   -released_at(F,T+1),
                   not terminated(F,T).

% (DEC6)
-holds_at(F,T+1) :- timepoint(T),
                    fluent(F),
                    -holds_at(F,T),
                    -released_at(F,T+1),
                    not initiated(F,T).

% (DEC7)
released_at(F,T+1) :- timepoint(T),
                      fluent(F),
                      released_at(F,T),
                      not initiated(F,T),
                      not terminated(F,T).
```

Task	DEC Axioms	Action	Location	Family Relation
1: Single supporting fact	✓	✓		
2: Two supporting facts	✓	✓		
3: Three supporting facts	✓	✓		
4: Two arg relations			✓	
5: Three arg relations		✓		
6: Yes/no questions	✓	✓		
7: Counting	✓	✓		
8: Lists/sets	✓	✓		
9 : Simple negation	✓	✓		
10: Indefinite knowledge	✓	✓		
11: Basic coreference	✓	✓		
12: Conjunction	✓	✓		
13: Compound coreference	✓	✓		
14: Time reasoning	✓	✓		
15: Basic deduction				
16: Basic induction				
17: Positional reasoning			✓	
18: Size reasoning				
19: Path finding	✓	✓	✓	
20: Agents motivations				
StepGame			✓	
gSCAN	✓		✓	
CLUTRR				✓
Pick&Place	✓	✓		

Table 9: Knowledge modules used for each of the tasks. Note that **DEC** Axioms, **action**, and **location** modules are used in at least two datasets. Some domains aren't listed as they are small and domain specific.

```
% (DEC8)
-released_at(F,T+1) :- timepoint(T),
                       fluent(F),
                       -released_at(F,T),
                       not released(F,T).

% (DEC9)
holds_at(F,T+1) :- timepoint(T),
                   fluent(F),
                   event(E),
                   happens(E,T),
                   initiates(E,F,T).

% (DEC10)
-holds_at(F,T+1) :- timepoint(T),
                    fluent(F),
                    event(E),
                    happens(E,T),
                    terminates(E,F,T).

% (DEC11)
released_at(F,T+1) :- timepoint(T),
                       fluent(F),
                       event(E),
                       happens(E,T),
                       releases(E,F,T).

% (DEC12)
-released_at(F,T+1) :- timepoint(T),
                        fluent(F),
                        event(E),
                        happens(E,T),
                        initiates(E,F,T).
-released_at(F,T+1) :- timepoint(T),
                        fluent(F),
                        event(E),
                        happens(E,T),
                        terminates(E,F,T).
```

E.2 Action Module

```
%********************
* common interface
* check: if location(unknown) is needed
********************%

% what happened in the given story
happens(action(A, pickup, I), T) :- pickup(A, I,
      T).
happens(action(A, drop, I), T) :- drop(A, I, T).
happens(action(A1, give, A2, I), T) :- give(A1,
     I, A2, T).
happens(action(A, goto, L), T) :- go(A, L, T).
happens(action(A, goto, L), T) :- isIn(A, L, T).

%********************
* basic atoms
********************%

direction(east; west; north; south).

agent(A) :- happens(action(A, _, _), _).
agent(A) :- happens(action(A, _, _, _), _).
agent(A) :- happens(action(_, give, A, _), _).
item(I) :- happens(action(_, pickup, I), _).
item(I) :- happens(action(_, drop, I), _).
item(I) :- happens(action(_, give, _, I), _).
location(L) :- happens(action(_, goto, L), _),
     not direction(L).

%********************
* atoms in DEC_AXIOMS
********************%
```

```
% event/1
event(action(A, pickup, I)) :- agent(A), item(I)
     .
event(action(A, drop, I)) :- agent(A), item(I).
event(action(A1, give, A2, I)) :- agent(A1),
     agent(A2), item(I), A1 != A2.
event(action(A, goto, L)) :- agent(A), location(
     L).
event(action(A, goto, D)) :- agent(A), direction
     (D).
event(action(robot, pick_and_place, Src, Dst))
     :- feature(Src, block), location(Dst), Src
     != Dst.

% timepoint/1
timepoint(T) :- happens(_, T). % the timepoint
     in story
timepoint(T) :- T=0..N, maxtime(N). % the
     timepoint for planning without story

% fluent/1
fluent(at(A, L)) :- agent(A), location(L).
fluent(at(I, L)) :- item(I), location(L).
fluent(carry(A, I)) :- agent(A), item(I).
fluent(on(B, L)) :- feature(B, block), location(
     L), B!=L.

% -released_at/2
%   1. -released_at(F, T) means commonsense law
     of inertia (CLI) can be applied to fluent F
     at T
%   2. CLI is also applied to this literal
     itself
-released_at(F, 0) :- fluent(F).

% holds_at/2
% initial states of fluents -- only location of
     items needs to be guessed
{holds_at(at(I, L), 0): location(L)} = 1 :- item
     (I).
holds_at(on(B, L), 0) :- on(B, L, 0).

% happens/2
% for each timepoint, at most 1 event happens;
     and it happens as fewer as possible
% {happens(E, T): event(E)}1 :- timepoint(T). %
     this rule would slow down many tasks
:~ happens(E, T). [1@0, E, T]

% every action should have some effect
%:- happens(E,T), not initiates(E,_,T).

% precondition on actions -- pickup
:- happens(action(A, pickup, I), T), holds_at(at
     (A, L1), T), holds_at(at(I, L2), T), L1 !=
     L2.

% initiates/3 and terminates/3

% effect of actions -- pickup
initiates(action(A, pickup, I), carry(A, I), T)
     :- agent(A), item(I), timepoint(T).

% effect of actions -- drop
terminates(action(A, drop, I), carry(A, I), T)
     :- agent(A), item(I), timepoint(T).

% effect of actions -- give
initiates(action(A1, give, A2, I), carry(A2, I),
      T) :- agent(A1), agent(A2), item(I),
     timepoint(T), A1 != A2.
terminates(action(A1, give, A2, I), carry(A1, I)
     , T) :- agent(A1), agent(A2), item(I),
     timepoint(T), A1 != A2.

% effect of actions -- goto
initiates(action(A, goto, L), at(A, L), T) :-
     agent(A), location(L), timepoint(T).
initiates(action(A, goto, L), at(I, L), T) :-
     holds_at(carry(A, I), T), location(L).
initiates(action(A, goto, D), at(A, L2), T) :-
    agent(A), location(L1), location(L2),
        timepoint(T),
    holds_at(at(A, L1), T), is(L2, D, L1).
terminates(action(A, goto, L1), at(A, L2), T) :-
      agent(A), location(L1), location(L2),
     timepoint(T), L1 != L2.
terminates(action(A, goto, L1), at(I, L2), T) :-
      holds_at(carry(A, I), T), location(L1),
     location(L2), L1 != L2.
terminates(action(A, goto, Direction), at(A, L),
      T) :-
    happens(action(A, goto, Direction), T),
    holds_at(at(A, L), T), Direction != L.

% effect of actions -- pick_and_place
initiates(action(robot, pick_and_place, Src, Dst
     ), on(Src, Dst), T) :-
    feature(Src, block), location(Dst), Src !=
        Dst, timepoint(T),
    not holds_at(on(_, Src), T),
    not holds_at(on(_, Dst), T): Dst!="table".

terminates(action(robot, pick_and_place, Src,
     Dst), on(Src, L), T) :-
    holds_at(on(Src, L), T), location(Dst), Dst
        != L.
```

E.3 Location Module

```
% general format translation, which can also be
     easily done in python script
% (this is not needed if we directly extract the
      general form in the beginning as in bAbI
     task4)
is(A, top, B) :- top(A, B).
is(A, top, B) :- up(A, B).
is(A, down, B) :- down(A, B).
is(A, left, B) :- left(A, B).
is(A, right, B) :- right(A, B).
is(A, top_left, B) :- top_left(A, B).
is(A, top_right, B) :- top_right(A, B).
is(A, down_left, B) :- down_left(A, B).
is(A, down_right, B) :- down_right(A, B).
is(A, east, B) :- east(A, B).
is(A, west, B) :- west(A, B).
is(A, south, B) :- south(A, B).
is(A, north, B) :- north(A, B).

% synonyms
synonyms(
    north, northOf; south, southOf; west, westOf
        ; east, eastOf;
    top, northOf; down, southOf; left, westOf;
        right, eastOf
).
synonyms(A, B) :- synonyms(B, A).
synonyms(A, C) :- synonyms(A, B), synonyms(B, C)
     , A!=C.
```

```
% define the offsets of 8 spacial relations
offset(
    overlap,0,0; top,0,1; down,0,-1; left,-1,0;
        right,1,0;
    top_left,-1,1; top_right,1,1; down_left
        ,-1,-1; down_right,1,-1
).

% derive the kind of spacial relation from
    synonyms and offset
is(A, R1, B) :- is(A, R2, B), synonyms(R1, R2).
is(A, R1, B) :- is(B, R2, A), offset(R2,X,Y),
    offset(R1,-X,-Y).

% derive the location of every object
% the search space of X or Y coordinate is
    within -100 and 100
% (to avoid infinite loop in clingo when data
    has error)
nums(-100..100).

location(A, Xa, Ya) :-
    location(B, Xb, Yb), nums(Xa), nums(Ya),
    is(A, Kind, B), offset(Kind, Dx, Dy),
    Xa-Xb=Dx, Ya-Yb=Dy.

location(B, Xb, Yb) :-
    location(A, Xa, Ya), nums(Xb), nums(Yb),
    is_on(A, Kind, B), offset(Kind, Dx, Dy),
    Xa-Xb=Dx, Ya-Yb=Dy.
```

E.4 Family Module

```
% gender

male(B) :- grandson(A, B).
male(B) :- son(A, B).
male(B) :- nephew(A, B).
male(B) :- brother(A, B).
male(B) :- father(A, B).
male(B) :- uncle(A, B).
male(B) :- grandfather(A, B).

female(B) :- granddaughter(A, B).
female(B) :- daughter(A, B).
female(B) :- niece(A, B).
female(B) :- sister(A, B).
female(B) :- mother(A, B).
female(B) :- aunt(A, B).
female(B) :- grandmother(A, B).

% gender-irrelevant relationships

sibling(A, B) :- siblings(A, B).
sibling(A, B) :- brother(A, B).
sibling(A, B) :- sister(A, B).
sibling(A, B) :- parent(A, C), parent(B, C), A
    != B.
sibling(A, B) :- sibling(B, A).
sibling(A, B) :- sibling(A, C), sibling(C, B), A
     != B.
sibling(A, B); sibling_in_law(A, B) :- child(A,
    C), uncle(C, B).
sibling(A, B); sibling_in_law(A, B) :- child(A,
    C), aunt(C, B).
sibling_in_law(A, B) :- sibling_in_law(B, A).
:- spouse(A, B), sibling(A, B).
:- spouse(A, B), sibling_in_law(A, B).
:- sibling(A, B), sibling_in_law(A, B).

spouse(A, B) :- wife(A, B).
spouse(A, B) :- husband(A, B).
spouse(A, B) :- spouse(B, A).

parent(A, B) :- father(A, B).
parent(A, B) :- mother(A, B).
parent(A, B) :- parent(A, C), spouse(C, B).
parent(A, B) :- sibling(A, C), parent(C, B).
parent(A, B) :- child(B, A).

child(A, B) :- children(A, B).
child(A, B) :- son(A, B).
child(A, B) :- daughter(A, B).
child(A, B) :- spouse(A, C), child(C, B).
child(A, B) :- child(A, C), sibling(C, B).
child(A, B) :- parent(B, A).

grandparent(A, B) :- grandfather(A, B).
grandparent(A, B) :- grandmother(A, B).
grandparent(A, B) :- parent(A, C), parent(C, B).
grandparent(A, B) :- grandchild(B, A).
grandparent(A, B) :- sibling(A, C), grandparent(
    C, B).
grandparent(A, B) :- grandparent(A, C), spouse(C
    , B).

grandchild(A, B) :- grandson(A, B).
grandchild(A, B) :- granddaughter(A, B).
grandchild(A, B) :- grandparent(B, A).

greatgrandparent(A, B) :- grandparent(A, C),
    parent(C, B).
greatgrandchild(A, B) :- greatgrandparent(B, A).

parent_in_law(A, B) :- spouse(A, C), parent(C, B
    ).
parent(A, B) :- spouse(A, C), parent_in_law(C, B
    ).
parent(A, B); parent_in_law(A, B) :- parent(C, A
    ), grandparent(C, B).
:- parent(A, B), parent(B, A).
:- parent(A, B), parent_in_law(A, B).
child_in_law(A, B) :- parent_in_law(B, A).

% gender-relevant relationships

greatgrandson(A, B) :- greatgrandchild(A, B),
    male(B).
greatgranddaughter(A, B) :- greatgrandchild(A, B
    ), female(B).

grandson(A, B) :- grandchild(A, B), male(B).
granddaughter(A, B) :- grandchild(A, B), female(
    B).

son(A, B) :- child(A, B), male(B).
daughter(A, B) :- child(A, B), female(B).
nephew(A, B) :- sibling(A, C), son(C, B).
niece(A, B) :- sibling(A, C), daughter(C, B).

husband(A, B) :- spouse(A, B), male(B).
wife(A, B) :- spouse(A, B), female(B).
brother(A, B) :- sibling(A, B), male(B).
sister(A, B) :- sibling(A, B), female(B).

father(A, B) :- parent(A, B), male(B).
mother(A, B) :- parent(A, B), female(B).
uncle(A, B) :- parent(A, C), brother(C, B).
```

```
aunt(A, B) :- parent(A, C), sister(C, B).

grandfather(A, B) :- grandparent(A, B), male(B).
grandmother(A, B) :- grandparent(A, B), female(B
    ).

greatgrandfather(A, B) :- greatgrandparent(A, B)
    , male(B).
greatgrandmother(A, B) :- greatgrandparent(A, B)
    , female(B).

son_in_law(A, B) :- child_in_law(A, B), male(B).
daughter_in_law(A, B) :- child_in_law(A, B),
    female(B).
father_in_law(A, B) :- parent_in_law(A, B), male
    (B).
mother_in_law(A, B) :- parent_in_law(A, B),
    female(B).
```

E.5 Domain Specific Modules

In this section, we list all domain-specific rules for each task. Some rules serve as an interface to turn the atoms in GPT-3 responses into a general format used in ASP modules. These rules are not necessary and can be removed if we let GPT-3 directly return the general atoms, e.g., "`query(at(A, where))`" instead of "`whereAgent(A)`". To save the cost for GPT-3 requests, we did not reproduce the experiments using new GPT-3 prompts with atoms in general formats.

E.5.1 bAbI Tasks 1 and 11

```
%%%% Interface -- these rules can be removed if
    we let GPT3 return the heads directly
query(at(A, where)) :- whereAgent(A).

% Find where last location of agent is
answer(L) :- query(at(A, where)), holds_at(at(A,
     L), T), T>=Tx: holds_at(at(A, _), Tx).
```

E.5.2 bAbI Task 2

```
%%%% Interface -- these rules can be removed if
    we let GPT3 return the heads directly
query(at(I, where)) :- loc(I).

% Find where last location of object is
answer(L) :- query(at(A, where)), holds_at(at(A,
     L), T), T>=Tx: holds_at(at(A, _), Tx).
```

E.5.3 bAbI Tasks 3 and 14

```
% the query before(O, L) is given, asking about
    the location of O before moving to L
% find all location changes of the queried
    object
location_change(L1, L2, T) :- before(O, _),
    holds_at(at(O, L1), T), holds_at(at(O, L2),
    T+1), L1 != L2.

% find the last location change to queried
    location
answer(L1) :- before(_, L2), location_change(L1,
     L2, T), T>=Tx: location_change(_, L2, Tx).
```

E.5.4 bAbI Task 4

```
answer(A) :- query(what, R1, B), is(A, R1, B).
answer(B) :- query(A, R1, what), is(A, R1, B).
```

E.5.5 bAbI Task 5

```
candidate(A1, T) :- query(action(who, give, A, I
    )), happens(action(A1, give, A2, I), T),
   A2=A: A!=anyone.
candidate(A2, T) :- query(action(A, give, who, I
    )), happens(action(A1, give, A2, I), T),
   A1=A: A!=anyone.
candidate(I, T) :- query(action(A1, give, A2,
    what)), happens(action(A1, give, A2, I), T).
location(unknown).

%%%% Interface -- these rules can be removed if
    we let GPT-3 return the heads directly
give(A1, A2, I, T) :- gave(A1, I, A2, T).
query(action(A1, give, A2, what)) :-
    whatWasGiven(A1, A2).
query(action(anyone, give, who, I)) :- received(
    I).
query(action(A1, give, who, I)) :- whoWasGiven(
    A1, I).
query(action(who, give, anyone, I)) :- whoGave(I
    ).
query(action(who, give, A2, I)) :- whoGave(I, A2
    ).

answer(A) :- candidate(A, T), Tx<=T: candidate(_
    , Tx).
```

E.5.6 bAbI Tasks 6 and 9

```
answer(yes) :- query(at(A, L)), holds_at(at(A, L
    ), T), Tx<=T: holds_at(at(A, _), Tx).
answer(no) :- not answer(yes).

%%%% Interface -- these rules can be removed if
    we let GPT-3 return the heads directly
query(at(A, L)) :- isIn(A, L).
```

E.5.7 bAbI Task 7

```
% find all items I that A is carrying at the
    last moment; then count I
carry(A, I) :- query(carry(A, count)), holds_at(
    carry(A,I),T),
   T>Tx: happens(E,Tx).
location(unknown).

%%%% Interface -- these rules can be removed if
    we let GPT-3 return the heads directly
query(carry(A, count)) :- howMany(A).

answer(N) :- query(carry(A, count)), N=#count{I:
     carry(A, I)}.
```

E.5.8 bAbI Task 8

```
%%%% Interface -- these rules can be removed if
    we let GPT-3 return the heads directly
query(carry(A, what)) :- carrying(A).
location(unknown).

% find all items I that A is carrying at the
    last moment
answer(I) :- query(carry(A, what)), holds_at(
    carry(A,I),T),
    T>Tx: happens(E,Tx).
```

E.5.9 bAbI Task 10

```
released(F,T) :- fluent(F), timepoint(T).

answer(yes) :- query(at(A, L)), holds_at(at(A, L
    ), T), Tx<=T: holds_at(at(A, _), Tx).

answer(maybe) :- query(at(A, L)), timepoint(T),
    1{isEither(A, L, _, T); isEither(A, _, L, T)
        },
    Tx<=T: holds_at(at(A, _), Tx);
    Tx<=T: isEither(A, _, _, Tx).

answer(no) :- not answer(yes), not answer(maybe)
    .

%%%% Interface -- these rules may be removed if
    we let GPT-3 return the heads directly
query(at(A, L)) :- isInQ(A, L).
holds_at(at(A, L), T) :- isIn(A, L, T).
go(A, L, T) :- move(A, L, T).
timepoint(T) :- isIn(_, _, T).
timepoint(T) :- isEither(_, _, _, T).
```

E.5.10 bAbI Tasks 12 and 13

```
%%%% Interface -- these rules can be removed if
    we let GPT-3 return the heads directly
query(at(A, where)) :- whereAgent(A).
go(A1, L, T) :- go(A1, A2, L, T).
go(A2, L, T) :- go(A1, A2, L, T).
```

E.5.11 bAbI Task 15

```
query(afraid(N, what)) :- agent_afraid(N).
```

E.5.12 bAbI Task 16

```
animal(frog;lion;swan;rhino).
color(green;white;yellow;gray).
isColor(Agent2,Color):- isAnimal(Agent,Animal),
    isColor(Agent,Color),isAnimal(Agent2,Animal)
    .
answer(Color) :- isColor(Name), isColor(Name,
    Color).
```

E.5.13 bAbI Task 17

```
% assume the 2nd queried object is at location
    (0,0)
location(B, 0, 0) :- query(_, _, B).

% the queried relation R is correct if its
    offset agrees with the location of A
answer(yes) :- query(A, R, B), offset(R, Dx, Dy)
    , location(A, X, Y),
    X>0: Dx=1; X<0: Dx=-1;
    Y>0: Dy=1; Y<0: Dy=-1.

answer(no) :- not answer(yes).

%%%% Interface -- these rules can be removed if
    we let GPT-3 return the heads directly
is(A, left, B) :- leftOf(A, B).
is(A, right, B) :- rightOf(A, B).
is(A, top, B) :- above(A, B).
is(A, down, B) :- below(A, B).
query(A, left, B) :- leftOf_nondirect(A, B).
query(A, right, B) :- rightOf_nondirect(A, B).
query(A, top, B) :- above_nondirect(A, B).
query(A, down, B) :- below_nondirect(A, B).
```

E.5.14 bAbI Task 18

```
smaller(A, B) :- bigger(B, A).
smaller(A, C) :- smaller(A, B), smaller(B, C).
answer(yes) :- query(smaller(A, B)), smaller(A,
    B).
answer(no) :- not answer(yes).

%%%% Interface -- these rules can be removed if
    we let GPT-3 return the heads directly
query(smaller(A, B)) :- doesFit(A, B).
query(smaller(A, B)) :- isBigger(B, A).
```

E.5.15 bAbI Task 19

```
agent(agent).
maxtime(10).
% location
location(L) :- is(L,_,_).
location(L) :- is(_,_,L).

% for each timestep, we take at most 1 action
{happens(action(agent, goto, D), T): direction(D
    )}1 :- timepoint(T).

% initial location
holds_at(at(agent, L), 0) :- initial_loc(L).

% goal
:- goal(L), not holds_at(at(agent, L), _).

% we aim to achieve the goal as early as
    possible
:~ goal(L), holds_at(at(agent, L), T). [-T@1,
    goal]
```

E.5.16 bAbI Task 20

```
loc(kitchen). loc(bedroom). loc(kitchen). loc(
    garden).
obj(pajamas). obj(football). obj(milk). obj(
    apple).

answer(Location) :- query(where, Agent, go), is(
    Agent, Quality), motivation(Quality,Location
    ), loc(Location).
```

```
answer(Quality) :- query(why, Agent, go,
     Location), is(Agent, Quality), motivation(
     Quality, Location), loc(Location).
answer(Quality) :- query(why,Agent, get, Obj),is
     (Agent, Quality), motivation(Quality, Obj),
     obj(Obj).
answer(Location) :- query(where, Agent, go), is(
     Agent, Quality), motivation(Quality,
     Location), loc(Location).
```

E.5.17 StepGame

```
% assume the 2nd queried object is at location
     (0,0)
location(Q2, 0, 0) :- query(_, Q2).

% extract answer relation R such that the offset
      (Ox,Oy) of R is in the same direction of (X
     ,Y)
answer(R) :- query(Q1, _), location(Q1, X, Y),
     offset(R, Ox, Oy),
    Ox=-1: X<0; Ox=0: X=0; Ox=1: X>0;
    Oy=-1: Y<0; Oy=0: Y=0; Oy=1: Y>0.
```

E.5.18 gSCAN

```
%********************
* find the goal
********************%

% features of objects
feature(O, shape, V) :- shape(O, V).
feature(O, color, V) :- color(O, V).
feature(O, size, V) :- size(O, V).

% feature of destination
feature(destination, V) :- query(walk),
     queryDesc(V).
feature(destination, V) :- query(push),
     queryDesc(V).
feature(destination, V) :- query(pull),
     queryDesc(V).

% find the destination object and location
pos_same(destination, O) :- feature(O,_,_),
    feature(O,_,V): feature(destination, V),
        feature(_,_,V).

same(destination, O) :- pos_same(destination, O)
     , feature(O, size, V),
    Vx<=V: feature(destination, big), pos_same(
        destination, Ox), feature(Ox, size, Vx);
    Vx>=V: feature(destination, small), pos_same
        (destination, Ox), feature(Ox, size, Vx)
        .

goal(at(agent,L)) :- same(destination, O), pos(O
     ,L).


%********************
* basic atoms
********************%
agent(agent).
item(I) :-  pos(I, L), I!=agent.
location((X,Y)) :- X=0..N-1, Y=0..N-1, gridSize(
     N).
```

```
is((X1,Y1), east, (X2,Y2)) :- location((X1,Y1)),
      location((X2,Y2)), X1=X2, Y1=Y2+1.
is((X1,Y1), west, (X2,Y2)) :- location((X1,Y1)),
      location((X2,Y2)), X1=X2, Y1=Y2-1.
is((X1,Y1), north, (X2,Y2)) :- location((X1,Y1))
     , location((X2,Y2)), X1=X2-1, Y1=Y2.
is((X1,Y1), south, (X2,Y2)) :- location((X1,Y1))
     , location((X2,Y2)), X1=X2+1, Y1=Y2.

pos_actions(walk; turn_left; turn_right; stay;
     push; pull).
left_dir(east, north; north, west; west, south;
     south, east).

%********************
* atoms in DEC_AXIOMS
********************%

% fluent/1
fluent(dir(A, L)) :- agent(A), direction(L).
fluent(ready(A)) :- agent(A).

% event/1
event(action(Agent, A)) :- agent(Agent),
     pos_actions(A).

% initial fluent values
holds_at(at(O,L),0) :- pos(O, L).
holds_at(dir(A,D),0) :- dir(A, D).

% for each timestep, we take at most 1 action
{happens(action(agent, A), T): pos_actions(A)}1
     :- timepoint(T).

% initial location
holds_at(at(agent, L), 0) :- initial_loc(L).

%%%%%%%%%%%%%%%
% action -- walk (to check simplification)
%%%%%%%%%%%%%%%

% initiates/3
initiates(action(A, walk), at(A, L2), T) :-
     agent(A), location(L), timepoint(T),
    holds_at(dir(A, D), T),
    holds_at(at(A, L1), T),
    is(L2, D, L1).

% terminates/3
terminates(action(A, walk), at(A, L1), T) :-
     agent(A), location(L), timepoint(T),
    holds_at(dir(A, D), T),
    holds_at(at(A, L1), T),
    is(L2, D, L1).

% precondition
% we don't walk in a deadend (i.e., the walk
     will result in no location change)
:- happens(action(agent, walk), T), not
     initiates(action(agent, walk), _, T).

%%%%%%%%%%%%%%%
% action -- turn_left (to check simplification)
%%%%%%%%%%%%%%%

% initiates/3
initiates(action(A, turn_left), dir(A, D2), T)
     :- agent(A), timepoint(T),
    holds_at(dir(A, D1), T),
    left_dir(D1, D2).
```

```
% terminates/3
terminates(action(A, turn_left), dir(A, D1), T)
     :- agent(A), timepoint(T),
    holds_at(dir(A, D1), T).

%%%%%%%%%%%%%%%
% action -- turn_right (to check simplification)
%%%%%%%%%%%%%%%

% initiates/3
initiates(action(A, turn_right), dir(A, D2), T)
     :- agent(A), timepoint(T),
    holds_at(dir(A, D1), T),
    left_dir(D2, D1).

% terminates/3
terminates(action(A, turn_right), dir(A, D), T)
     :- agent(A), timepoint(T),
    holds_at(dir(A, D), T).

%%%%%%%%%%%%%%%
% action -- push/pull
%%%%%%%%%%%%%%%

% initiates/3 for objects with size <= 2
initiates(action(A, push), at(A, L2), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T),
    same(destination, Target), holds_at(at(
        Target, L1), T),
    is(L2, D, L1), feature(Target, size, V), V
        <= 2.

initiates(action(A, push), at(Target, L2), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T),
    same(destination, Target), holds_at(at(
        Target, L1), T),
    is(L2, D, L1), feature(Target, size, V), V
        <= 2.

initiates(action(A, pull), at(A, L2), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T),
    same(destination, Target), holds_at(at(
        Target, L1), T),
    is(L1, D, L2), feature(Target, size, V), V
        <= 2.

initiates(action(A, pull), at(Target, L2), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T),
    same(destination, Target), holds_at(at(
        Target, L1), T),
    is(L1, D, L2), feature(Target, size, V), V
        <= 2.

% terminates/3 for objects with size <= 2
terminates(action(A, push), at(A, L1), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T),
    same(destination, Target), holds_at(at(
        Target, L1), T),
    is(L2, D, L1), feature(Target, size, V), V
        <= 2.

terminates(action(A, push), at(Target, L1), T)
     :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T),
    same(destination, Target), holds_at(at(
        Target, L1), T),
    is(L2, D, L1), feature(Target, size, V), V
        <= 2.

terminates(action(A, pull), at(A, L1), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T),
    same(destination, Target), holds_at(at(
        Target, L1), T),
    is(L1, D, L2), feature(Target, size, V), V
        <= 2.

terminates(action(A, pull), at(Target, L1), T)
     :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T),
    same(destination, Target), holds_at(at(
        Target, L1), T),
    is(L1, D, L2), feature(Target, size, V), V
        <= 2.

% initiates/3 for objects with size >= 3

initiates(action(A, push), ready(A), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T), not holds_at(ready(A), T)
        ,
    same(destination, Target), holds_at(at(
        Target, L1), T), feature(Target, size, V
        ), V >= 3.

initiates(action(A, push), at(A, L2), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T), holds_at(ready(A), T),
    same(destination, Target), holds_at(at(
        Target, L1), T), feature(Target, size, V
        ), V >= 3,
    is(L2, D, L1).

initiates(action(A, push), at(Target, L2), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T), holds_at(ready(A), T),
    same(destination, Target), holds_at(at(
        Target, L1), T), feature(Target, size, V
        ), V >= 3,
    is(L2, D, L1).

initiates(action(A, pull), ready(A), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T), not holds_at(ready(A), T)
        ,
    same(destination, Target), holds_at(at(
        Target, L1), T), feature(Target, size, V
        ), V >= 3.

initiates(action(A, pull), at(A, L2), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T), holds_at(ready(A), T),
    same(destination, Target), holds_at(at(
        Target, L1), T), feature(Target, size, V
        ), V >= 3,
    is(L1, D, L2).

initiates(action(A, pull), at(Target, L2), T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T), holds_at(ready(A), T),
    same(destination, Target), holds_at(at(
        Target, L1), T), feature(Target, size, V
        ), V >= 3,
    is(L1, D, L2).

% terminates/3 for objects with size >= 3

{ terminates(action(A, push), ready(A), T);
  terminates(action(A, push), at(A, L1), T);
  terminates(action(A, push), at(Target, L1), T)
}=3 :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T), holds_at(ready(A), T),
    same(destination, Target), holds_at(at(
        Target, L1), T), feature(Target, size, V
        ), V >= 3,
    is(L2, D, L1).

{ terminates(action(A, pull), ready(A), T);
  terminates(action(A, pull), at(A, L1), T);
  terminates(action(A, pull), at(Target, L1), T)
}=3 :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T), holds_at(ready(A), T),
    same(destination, Target), holds_at(at(
        Target, L1), T), feature(Target, size, V
        ), V >= 3,
    is(L1, D, L2).

% precondition

% 1. we don't push/pull in a deadend (i.e., the
     action will result in no location change)
:- happens(action(agent, push), T), not
     initiates(action(agent, push), _, T).
:- happens(action(agent, pull), T), not
     initiates(action(agent, pull), _, T).

% 2. the agent can push/pull only if it's
     queried
:- happens(action(agent, push), _), not query(
     push).
:- happens(action(agent, pull), _), not query(
     pull).

% 2. it's not allowed to have 3 objects (agent +
      2 items) in the same cell
% (I use holds_at(_, T) instead of timepoint(T)
     since the latter doesn't cover the last T+1
     timestamp)
%:- holds_at(_, T), location(L), N = #count{O:
     holds_at(at(O, L), T)}, N>2.

% 3. after push/pull, the agent cannot do a
     different action in {walk, push, pull}
:- happens(action(agent, A1), T1), happens(
     action(agent, A2), T2), A1!=A2, T1<T2,
    1{A1=push; A1=pull},
    1{A2=push; A2=pull; A2=walk}.

% 4. the agent cannot change its direction to
     push/pull after reaching destination
reach_destination(T) :- goal(at(agent,L)),
     holds_at(at(agent, L), T),
    not reach_destination(Tx): timepoint(Tx), Tx
        <T.
:- reach_destination(T1), holds_at(dir(agent, D1
     ), T1),
    holds_at(dir(agent, D2), T2), happens(action
         (agent, push), T2), T1<T2, D1!=D2.
```

```
:- reach_destination(T1), holds_at(dir(agent, D1
     ), T1),
    holds_at(dir(agent, D2), T2), happens(action
         (agent, pull), T2), T1<T2, D1!=D2.

%%%%%%%%%%%%%%%
% goal
%%%%%%%%%%%%%%%

% 1. (optional to speed up) we need to reach the
      destination and as early as possible
:- goal(at(agent,L)), not reach_destination(_).
:~ goal(at(agent, L)), reach_destination(T). [
     T@10, goal]

% 2. we need to reach the goal and as early as
     possible
%   a. the direction when reaching goal must
     align with the direction when reaching
     destination
%   b. if it's not deadend, there must be
     something blocking the next push/pull
reach_goal(T) :-
    agent(A), holds_at(at(A, L1), T), holds_at(
        dir(A, D), T),
    same(destination, Target), holds_at(at(
        Target, L1), T),
    reach_destination(Tr), holds_at(dir(A, D),
        Tr),
    holds_at(at(_, L2), T): query(push), is(L2,
        D, L1);
    holds_at(at(_, L2), T): query(pull), is(L1,
        D, L2);
    not reach_goal(Tx): timepoint(Tx), Tx<T.
:- not reach_goal(_).
:~ reach_goal(T). [T@9, goal]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% additional requirements to achieve the goal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% the agent cannot move further before reaching
     destination
:- reach_destination(T), goal(at(agent, (Xg,Yg))
     ),
    holds_at(at(agent, (X1,Y1)), Tx), holds_at(
        at(agent, (X2,Y2)), Tx+1), Tx<T,
    |X1-Xg| + |Y1-Yg| < |X2-Xg| + |Y2-Yg|.

% by default, walking all the way horizontally
     first and then vertically
move(horizontally, T) :- happens(action(agent,
     walk), T), holds_at(dir(agent, D), T), 1{D=
     east; D=west}.
move(vertically, T) :- happens(action(agent,
     walk), T), holds_at(dir(agent, D), T), 1{D=
     south; D=north}.
:- not while(zigzagging), move(horizontally, T1)
     , move(vertically, T2), T1>T2.

% hesitantly: the agent must stay after every
     action in {walk, push, pull}
:- while(hesitantly), happens(action(agent, A),
     T),
    1{A=walk; A=push; A=pull},
    not happens(action(agent, stay), T+1).

% cautiously
cautious(T) :- happens(action(agent, turn_left),
      T),
    happens(action(agent, turn_right), T+1),
    happens(action(agent, turn_right), T+2),
    happens(action(agent, turn_left), T+3).

% the agent must be cautious before every action
     in {walk, push, pull}
:- while(cautiously), happens(action(agent, A),
    T),
    1{A=walk; A=push; A=pull},
    not cautious(T-4).

% spinning
spin(T) :- happens(action(agent, turn_left), T),
    happens(action(agent, turn_left), T+1),
    happens(action(agent, turn_left), T+2),
    happens(action(agent, turn_left), T+3).

% we always spin at the beginning if there is
    any action
:- while(spinning), happens(_,_), not spin(0).

% we always spin after every action in {walk,
    push, pull} except for the last one
:- while(spinning), happens(action(agent, A1),
    T1), happens(action(agent, A2), T2), T1<T2,
    1{A1=walk; A1=push; A1=pull},
    1{A2=walk; A2=push; A2=pull},
    not spin(T1+1).

% zigzagging
% if horizontal move is needed, the first move
    must be horizontal
:- while(zigzagging), move(horizontally, _),
    move(D, Tmin), D!=horizontally,
    Tmin<=Tx: move(_,Tx).
% if a different kind of move D2 is after D1, D2
     must be followed directly
:- while(zigzagging), move(D1, T1), move(D2, T2)
    , D1!=D2, T1<T2,
    not move(D2, T1+2).
```

E.5.19 Pick&Place

```
%%%%%
% Set up the environment
%%%%%

% Define the number of grippers for the robot
#const grippers=1.

% Define the maximum number of steps to consider
{maxtime(M): M=0..10} = 1.
:~ maxtime(M). [M]

%%%%%
% Extract the features for all items in the
    intial and goal states
% we assume these items form the complete set of
     items in this example
%%%%%

feature(I, F) :- on(I,_), F=@gen_feature(I).
feature(I, F) :- on(I,_,0), F=@gen_feature(I).
feature(I, F) :- on(_,I), I!="table", F=
    @gen_feature(I).
feature(I, F) :- on(_,I,0), I!="table", F=
    @gen_feature(I).

% Define all locations
location("table").
location(L) :- feature(L, block).
location(L) :- feature(L, bowl).

%********************
* atoms in DEC_AXIOMS
********************%

% happens/2
{happens(E,T): event(E)}grippers :- timepoint(T)
    .

% **** constraints ****

% the goal must be achieved in the end
:- maxtime(M), on(A, B), not holds_at(on(A,B), M
    +1).

% At any time T, for each block/bowl, there
    cannot be 2 items directly on it
:- timepoint(T), feature(L, _), 2{holds_at(on(I,
     L), T): feature(I,_)}.

% if there are bowls on the table, a block can
    only be on a block or a bowl;
:- feature(_,bowl), feature(I,block), holds_at(
    on(I,L),_), {feature(L, block); feature(L,
    bowl)} = 0.

% there cannot be more than max_height-1 blocks
    stacked on a block
up(A,B,T) :- holds_at(on(A, B), T).
up(A,C,T) :- up(A,B,T), up(B,C,T).
:- timepoint(T), feature(L, block), #count{I: up
     (I,L,T)} >= max_height.
```

F Dataset Errors

This section enumerates the errors in the datasets we found.

F.1 bAbI

In task 5, the dataset has two errors with regard to the labels.

Error #1. In the following example, the answer is ambiguous since Bill gives Mary both the football and the apple.

```
CONTEXT:
Mary journeyed to the kitchen.
Mary went to the bedroom.
Mary moved to the bathroom.
Mary grabbed the football there.
Mary moved to the garden.
Mary dropped the football.
Fred went back to the kitchen.
Jeff went back to the office.
Jeff went to the bathroom.
Bill took the apple there.
Mary picked up the milk there.
Mary picked up the football there.
Bill went back to the kitchen.
Bill went back to the hallway.
Fred journeyed to the office.
Bill discarded the apple.
Mary journeyed to the kitchen.
```

```
Fred journeyed to the garden.
Mary went to the hallway.
Mary gave the football to Bill.
Bill passed the football to Mary.
Bill took the apple there.
Bill gave the apple to Mary.
Jeff travelled to the kitchen.

QUERY:
What did Bill give to Mary?

PREDICTION:
apple

Answer:
football
```

Error #2. In the following example, the answer is ambiguous since Fred gives Bill both the milk and the apple.

```
CONTEXT:
Mary journeyed to the bathroom.
Mary moved to the hallway.
Mary went to the kitchen.
Bill went back to the bedroom.
Bill grabbed the apple there.
Fred went back to the garden.
Mary went to the garden.
Fred took the milk there.
Jeff moved to the hallway.
Bill dropped the apple there.
Fred handed the milk to Mary.
Mary handed the milk to Fred.
Fred went back to the bedroom.
Fred passed the milk to Bill.
Fred took the apple there.
Fred gave the apple to Bill.
Jeff went to the kitchen.
Bill dropped the milk.

QUERY:
What did Fred give to Bill?

PREDICTION:
apple

Answer:
milk
```

F.2 CLUTRR

We detected 16 data errors in the CLUTRR 1.3 dataset using our method. These errors can be grouped into the following 4 categories.

- 5 data instances are due to incorrect relation graphs. For example, one relation graph contains the main part “A-son-B-daughter-C-aunt-D” and a noise (supporting) relation “B-spouse-D”. However, if B and D are couples, then C should have mother D instead of aunt D.
- 9 data instances have a correct relation graph (e.g., A-son-B-grandmother-C-brother-D with a noise supporting relation B-mother-A) but the noise relation is translated into a sentence with a wrong person name (e.g., "D has mother A" instead of "B has mother A").
- 1 data instance has a correct relation graph and story, but has a wrong label (i.e., the label should be mother_in_law instead of mother).
- 1 data instance has a correct relation graph and story, but the query cannot be answered due to the ambiguity of a sentence. It uses "A has grandsons B and C" to represent brother(B, C), while B and C may have different parents.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section 7.

□ A2. Did you discuss any potential risks of your work?
Not applicable. Left blank.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Section 1.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Sections 2, 3, 4.

☑ B1. Did you cite the creators of artifacts you used?
Sections 2, 3, 4.

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
At the beginning of the appendix.

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Section 4 and Appendix A, B.

□ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Section 4.

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Section 4.

C ☑ Did you run computational experiments?

Section 4.

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
At the beginning of the appendix.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 4.

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Section 4.

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 4.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

□ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

□ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

□ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

□ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

□ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.