Using Learning from Answer Sets for Robust Question Answering with LLM

Irfan Kareem[1](✉), Katie Gallagher[2], Manuel Borroto[1], Francesco Ricca[1], and Alessandra Russo[3]

[1] Department of Mathematics and Computer Science, University of Calabria, Cosenza, Italy
{irfan.kareem,manuel.borroto,francesco.ricca}@unical.it

[2] The University of Chicago, Chicago, USA
krgallagher@uchicago.edu

[3] Department of Computing, Imperial College London, London, UK
a.russo@imperial.ac.uk

Abstract. Large Language Models (LLMs) lack the ability for commonsense reasoning and learning from text. In this work, we present a system, called LLM2LAS, for learning commonsense knowledge from story-based question and answering expressed in natural language. LLM2LAS combines the semantic parsing capability of LLMs with ILASP for learning commonsense knowledge expressed as answer set programs. LLM2LAS requires only few examples of questions and answers to learn general commonsense knowledge and correctly answer unseen questions. An empirical evaluation demonstrates the viability of our approach.

Keywords: Logic-based Learning · Knowledge Representation · Question and Answering (Q&A)

1 Introduction

A challenging problem in Artificial Intelligence (AI) is the ability to endow machines with commonsense reasoning and learning capabilities [4]. In machine comprehension and question and answering (Q&A) tasks, AI models, trained to answer questions about given texts, may do so through learning correlations and statistical features in the text rather than learning commonsense knowledge and/or performing general commonsense reasoning [1]. Despite recent successes, AI approaches such as Large Language Models (LLMs), have also shown to fall short in providing truthful answers [22], and to under perform on natural language reasoning benchmarks [10,19]. Their lack of transparency and explainability make it difficult to ascertain whether these models genuinely learn commonsense reasoning [17]. On the other end, LLMs have shown great potential in processing and generating text. In particular, they have shown to be effective

This work was equally conceived by M. Borroto, K. Gallagher and I. Kareem.

in performing semantic parsing of natural language, the task of mapping natural language utterances into formal meaning representations [5]. Recent neuro-symbolic approaches have started to show that by combining LLMs and formal reasoning methods it is possible to address some of the limitations of LLMs whilst exploiting their generative capability. LLM's coherence and consistency in story completion tasks can be improved by combining LLM-based semantic parsing, to map text into formal representations, with symbolic reasoner to determine which of the LLM's generated completion sentences are correct [14]. Ishay et al. combine LLMs with answer set programming (ASP) [12] to solve logic puzzles [7]. Yang et al. combines GPT3-based semantic parsing with an ASP knowledge module to perform reasoning, showing state-of-the-art performance on several benchmarks [21]. Although these approaches result in more robust reasoning from text, the symbolic knowledge of the reasoning component is manually engineered, which is a time consuming task.

In this work we propose a new system, called LLM2LAS, that learns common-sense knowledge for machine comprehension. LLM2LAS combines an LLM-based semantic parser for generating symbolic representations of a story and questions, with ILASP, a tool for learning common-sense knowledge under the answer set semantics. Our approach automatically generates a ILASP learning task from a given story, questions and answers, written in natural language, and iteratively learns the common-sense knowledge needed for solving the given Q&A task. The general learned knowledge can then be used to answer questions about unseen texts. The key components of LLM2LAS include (i) an open-source LLM-based few-shot semantic parser for generating ASP representations from natural language; (ii) a learning module, based on ILASP, for learning common-sense knowledge needed to answer questions about given stories, and (iii) a reasoning module for answering questions about a story using the learned common-sense knowledge. We have evaluated our approach on the Q&A bAbI dataset [15], reporting 100% accuracy on 13 of the 20 bAbI tasks.

2 Related Work

Mitra et al. developed a three-layer Q&A system that combines statistical methods with inductive rule learning and reasoning [13]. The system includes a Statistical Inference layer, which uses an Abstract Meaning Representation (AMR) parser, a Translation layer, which converts the AMR parser output into Event Calculus syntax using a naive deterministic algorithm, and the Reasoner layer, which uses a modified version of the ILP algorithm XHAIL [16] to learn the knowledge required for reasoning. The system achieves on the bAbI dataset an accuracy of 99.68%, but requiring users to manually specify mode declarations and task-dependent background knowledge.

Nye et al. proposed a neuro-symbolic approach to improve coherence and consistency of text generation in a story completion task [14]. The approach uses GPT-3 to generate candidate completion sentences and an LLM-based parser to derive logical representations of a given story and generated sentences. The latter are compared to symbolic candidates inferred using a minimal world model to check consistency. Only consistent candidates are considered for the final generation. The system performs well on different benchmarks e.g. bAbI, CLUTRR, and gSCAN. However, the main limitation is the manual engineering of the world model, which is task specific.

Ishay et al. combined LLM and ASP to solve logical puzzles in a step-by-step manner [7]. The method uses GPT-3 with prompt engineering to extract relevant objects, their categories and typed predicates from text descriptions of the puzzles. It then generates an ASP program that captures the rules of the given puzzle, using a Generate-Define-Test approach. The outcomes are computed symbolically using the generated ASP program. The method is interpretable but requires human intervention to resolve errors in the generation process.

Yang et al. demonstrated GPT-3 to be effective in few-shot semantic parsing of natural language into ASP representation [21]. Their approach handles Q&A tasks but with task-specific manually handcrafted background knowledge, achieving promising results on different NLP benchmarks including bAbI, StepGame, CLUTRR and gSCAN. Our approach differs from this work in that we can learn the relevant knowledge needed to solve a task.

We adopt Learning from Answer Set (LAS) to learn the knowledge needed to solve a Q&A task, thus reducing human intervention, and exploit LLM-based semantic parsing capability to automatically generate LAS learning tasks from the given natural language dataset. This combination of LLM and LAS is novel and offers promising performance.

3 Preliminaries

Answer Set Programming (ASP) is a well-known paradigm for specifying real-world problems, common-sense knowledge and solving combinatorial optimisation problems [2,6]. We provide here a brief recap of the ASP syntax relevant to this paper, referring the reader to [2,3,6] for a formal account on ASP syntax and semantics.

Given atoms $\mathtt{h}$, $\mathtt{b}_1, \ldots, \mathtt{b}_n$, $\mathtt{c}_1, \ldots, \mathtt{c}_m$, a *normal rule* is of the form $\mathtt{h} \texttt{:-} \mathtt{b}_1, \ldots, \mathtt{b}_n, \texttt{ not } \mathtt{c}_1, \ldots, \texttt{ not } \mathtt{c}_m$, where $\mathtt{h}$ is the *head*, $\mathtt{b}_1, \ldots, \mathtt{b}_n, \texttt{ not } \mathtt{c}_1, \ldots, \texttt{ not } \mathtt{c}_m$ (collectively) is the *body* of the rule, and "`not`" represents negation as failure. Rules $\texttt{:-}\ \mathtt{b}_1, \ldots, \mathtt{b}_n, \texttt{ not } \mathtt{c}_1, \ldots, \texttt{ not } \mathtt{c}_m$ are called *hard constraints*. ASP programs include also *choice rules*. A choice rule is a special type of rule of the form $\mathtt{l}\{\mathtt{h}_1, \ldots, \mathtt{h}_k\}\mathtt{u} \texttt{:-}\ \mathtt{b}_1, \ldots, \mathtt{b}_n, \texttt{ not } \mathtt{c}_1, \ldots, \texttt{ not } \mathtt{c}_m$, where $\mathtt{l}$ and $\mathtt{u}$ are integers. The semantics of ASP programs is in terms of stable models (or answer sets) [6].

To represent and reasoning about events, a known formalism is the Event Calculus (EC) [9]. EC is a normal logic program that typically includes domain-independent rules describing general principles for inferring when properties (called *fluents*) are true at particular time-points, denoted as `holdsAT(F, T)`, based on which events have previously occurred (denoted as *happens(E, T)*). In addition, an EC representation includes a collection of *domain-dependent* rules, describing the effect of events (using the predicates `initiatedAt(F, T)` and terminatedAt(F, T)) as well as the time point at which events occur (using the predicate happensAt(E, T)). In this paper we will make use of the following ASP encoding of domain-independent rules for EC:

$$\begin{array}{l}\texttt{holdsAt(F,T+1):-initiatedAt(F,T),time(T).}\\ \texttt{holdsAt(F,T+1):-holdsAt(F,T),not terminatedAt(F,T),time(T).}\end{array} \tag{1}$$

where initiatedAt(F, T) (resp. terminatedAt(F, T)) indicates the time T when a fluent F is made true (resp. false) by an event occurring. In learning commonsense knowledge we may observe fluents at particular time points, encoded using holdsAt predicates with the objective of learning the definition of initiatedAt and terminatedAt, that is how events cause changes in the truth value of fluents.

Learning from Answer Sets. In this paper, we use *Learning from Answer Sets* (LAS) framework and its state-of-the-art system ILASP [11] for learning ASP programs. The LAS framework solves *learning tasks* which consist of a *background knowledge*, the *mode bias* and a set of *examples*. The background knowledge, denoted as B, is an ASP program which describes a set of concepts that are known before learning. The mode bias, denoted as M and often called *language bias*, is used to express the ASP programs that can be learned. A *mode bias* is defined as a pair of sets of mode declarations $M = \langle M_h, M_b \rangle$, where M_h (resp. M_b) are called the *head* (resp. *body*) *mode declarations*. Each mode declaration is a literal whose abstracted arguments are either var(t) or const(t), for some constant t (called a *type*). For each type, a set of constants is provided along with the maximum number of variables that a rule can take, thus constraining the search space induced M. Informally, a literal is *compatible* with a mode declaration m if it can be constructed by replacing every instance of var(t) in m with a variable of type t, and every const(t) with a constant of type t.

Definition 1. *Given a mode bias $M = \langle M_h, M_b \rangle$, a normal rule R is in the hypothesis space S_M if and only if (i) the head of R is compatible with a mode declaration in M_h; (ii) each body literal of R is compatible with a mode declaration in M_b; and (iii) no variable occurs with two different types.*

The set of examples, denoted as E, describes a set of semantic properties that the learned ASP program should satisfy. They are defined in terms of *partial interpretations*. A partial interpretation is a pair of sets of ground atoms $\langle e^{inc}, e^{exc} \rangle$, called respectively inclusion and exclusion sets. An interpretation I *extends* e iff $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$. A ILASP example $e \in E$ is a *context dependent partial interpretation* (CDPI). This is a tuple $e = \langle e_{id}, e_{pi}, e_{ctx} \rangle$, where e_{id} is an identifier for e, e_{pi} is a partial interpretation and e_{ctx} is an ASP program called a *context*. A CDPI e is *accepted* by a program P if and only if there is an answer set of $P \cup e_{ctx}$ that extends e_{pi}. The idea of a context-dependent example is that each context only applies to a particular example. This is suitable for our question-answering tasks where answer to a question is normally contextualised with respect to the story, or text provided to the learner. Formally, an ILASP *context-dependent learning task* is defined as follows.

Definition 2. *A Context-dependent Learning task ($ILP_{LAS}^{context}$) is a tuple $T = \langle B, S_M, E\rangle$ where B is an ASP program, called the background knowledge, S_M is the set of rules allowed in the hypotheses (the hypothesis space), E is a set of CDPIs. A hypothesis H is an inductive solution of T (written $H \in ILP_{LAS}^{context}(T)$) if and only if:*

1. *$H \subseteq S_M$;*
2. *$\forall\langle e_{id}, e_{pi}, e_{ctx}\rangle \in E$, $\exists A \in AS(B \cup e_{ctx} \cup H)$ such that A extends e_{pi}*

A learning task may have multiple inductive solutions. These are scored in terms of their length (i.e., number of literals they include), $score(H, T) = |H|$. An inductive solution $H \in ILP_{LAS}^{context}(T)$ is *optimal* if there is no other inductive solution $H' \in ILP_{LAS}^{context}(T)$ such that $score(H', T) < score(H, T)$.

Large Language Models and POS Tagging. The introduction of LLM models, such as GPT and BERT, has revolutionized Natural Language Processing (NLP) by enabling machines to process and generate human language with unprecedented accuracy [18]. These deep neural network models owe their effectiveness to the transformer-based architectures [18], which utilize self-attention mechanisms to process and contextualize vast amounts of text. Most currently available LLMs have billions of parameters and are trained in a self-supervised way to predict missing tokens or the next token in a given sequence. LLMs are usually instructed through text prompts to solve a specific task, such as translating or answering questions. They have also been used successfully for semantic parsing, i.e., converting text into a structured format for analysis [5,14,21].

Part-of-speech (POS) tagging involves assigning labels to tokens within a text based on their grammatical function, i.e., whether the token is a noun, verb, adjective, adverb, or other [8]. Given a sequence $x_1, x_2, \ldots, x_n$ of words (tokens) and a set of tags, the task is to generate a sequence $y_1, y_2, \ldots, y_n$ of tags, where y_i represents the assigned tag for the input x_i. POS tagging presents challenges due to word ambiguities because a word can have multiple meanings and functions depending on the context in which it is used. Current POS tagging techniques appeal to the text context to address the ambiguity issue. In our approach, we use Universal POS tags by leveraging the spaCy library (https://spacy.io/).

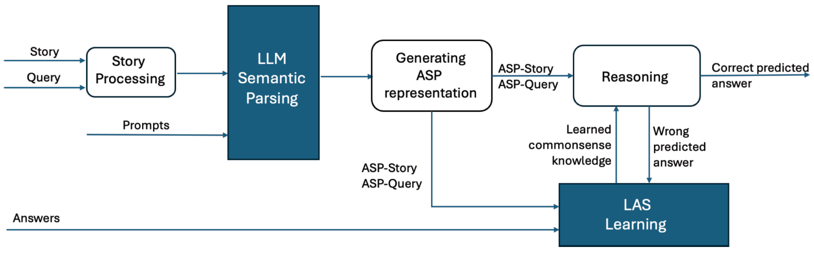

Fig. 1. Architecture of LLM2LAS

4 Methodology

In this section, we present our neuro-symbolic system LLM2LAS, which combines LLMs with LAS to learn commonsense knowledge for story-based Q&A expressed in natural language. As illustrated in Fig. 1, the system consists of several modules. The *Story Processing* normalizes the story statements and enriches them with POS tagging data; the *LLM Semantic Parsing* generates from the given story relevant fluent representations. These are then used to generate ASP representation of the narrative described in the story. The reasoner module attempts to answer the question using the extracted narrative and the domain-independent rules given in (1). If the answer is incorrect, the learner module is invoked to learn relevant commonsense knowledge from the given narrative, question and ground truth answer. We detail each of these steps in what follows.

Story Processing. The module receives as input a story and a question from which the system is supposed to learn some knowledge. A story consists of an ordered set of statements describing a narrative or a scenario, while the question is designed to be answered by exploiting the information in the story. Each question is associated with the correct and incorrect answers. All sentences are *normalized* by identifying coreferences, both basic and compound, in the text and replacing them with the corresponding referents. The coreference resolution task determines whether two different referring expressions point to the same entity. For example, in basic coreference the sentence *"Mary went to the store, and she bought food."*, the word *"she"* is replaced by *"Mary"*. Moreover, sentences containing negations are identified and flagged to support the automated generation of mode bias in the learning phase. Finally, the module enriches each sentence in the story with a POS tagging sequence and a syntactic parse tree. To perform these pre-processing tasks, we rely on the NLP library spaCy.

LLM Semantic Parsing. LLMs have proven to work well in many NLP tasks, including semantic parsing [5,14,21]. We exploit this strength and leverage an LLM to parse the processed sentences and the question into fluent representation, i.e., a predicate that encodes the semantic meaning of a sentence within a story and illustrates the facts. The fluent representation allows for creating the EC representations and the mode bias declarations in the next stage.

Most available LLMs are trained on extensive public data, allowing them to achieve reasonable zero-shot generalization on diverse tasks. However, these models are not expected to perform as well in domain-specific semantic parsing tasks, where the inductive bias from pre-training is less favorable. To address this limitation, we used the few-shot prompting technique, which involves giving the model a few task-specific examples within the prompt to help guide its responses [5]. Listing 1.1 shows an example of the prompt we have designed to ask the LLM to parse the bAbI dataset statements.

For example, if we ask an LLM model to parse the sentence *"Sam moved to the bathroom."*, using the previous prompt, the result would be: `go_to(sam, bathroom)`. Our system parses each statement separately, using the

Listing 1.1. Prompt for Fact Extraction

```
Please parse the following English sentences into the semantic parse.
The available keywords
are: go_to, and be_in:
Sentence: Mary moved to the bathroom.
Semantic parse: go_to(mary,bathroom)
Sentence: John went to the hallway.
Semantic parse: go_to(john,hallway)
...
Sentence: Where is Sandra?
Semantic parse: be_in(sandra,V1)

Your turn:
```

same prompt multiple times to give a precise semantic representation in fluent terms. Table 1 (second column) shows a few examples of fluent representations.

Generating ASP Representation. Once statements of the story are parsed into their corresponding fluent representation, the next step is to create the EC representations (if needed) and the mode bias fluent. The EC representation depicts the actions and their effects in the story and comprises the four predicates introduced in Sect. 3. On the other hand, the mode bias fluents aid the learner in automatically generating mode bias declarations for both formal representations.

The construction of the EC representation from the fluent representation involves choosing an EC predicate and a time point. Given a sentence and its fluent representation, we select the predicate according to the following schema: (i) if the sentence is a question, then the "holdsAt" predicate is used; (ii) if the base of the literal's predicate is "be" and the statement is negated, then the "terminatedAt" predicate is used; (iii) if the base of the literal's predicate is "be" and the statement is not negated, then the "initiatedAt" predicate is used; (iv) otherwise, the "happensAt" predicate is used. The time point for the EC predicate is determined by the sentence's placement within the story. The first sentence is given time point 1, and every subsequent sentence has a time point determined by the previous one plus 1. Questions are given a time point according to when they are asked. Table 1 shows some examples of statements and their representations.

Table 1. Examples of statements with fluent, and EC representations (Rep.).

Statement	Fluent Rep.	Event Calculus Rep.
Mary went to the garden.	go_to(mary, garden)	happensAt(go_to(mary, garden), T)
John and Helen went to the store.	go_to(john, store), go_to(helen, store)	happensAt(go_to(john, store), T), happensAt(go_to(helen, store), T)
John is in the park or garden.	{be_in(john, park), be_in(john, garden)}	{initiatedAt(be_in(john, park), T), initiatedAt(be_in(john, garden), T)}
Yesterday Ana went to the park.	go_to(ana, park, yesterday)	happensAt(go_to(ana, park, yesterday), T)

Mode bias fluents consist of atoms from the sentence's fluent representation where all arguments have been replaced by their types wrapped in either *"var"* or *"const"*.The argument types are determined using the POS tagging data and the

WH-determiners of the questions. If the sentence fluent contains an argument that is a variable (i.e., the sentence is a WH-question), then the variable is given a type in the following way: if the sentence is a *"what"*, *"when"*, or *"where"* question, then the variable's type is *"nn"*; if the sentence is a *"who"* question, then the variable's type is *"nnp"*; if the sentence is a *"why"* question, then the variable's type is *"jj"*; if the sentence is a *"how many"* question, then the variable's type is *"number"*. In all other cases, the argument's type is given by its associated POS tag. We provide the types for all arguments that have a temporal aspect and the types of variables in *"why"* questions with *"const"* wrappings. The types of all other arguments are given *"var"* wrappings. Table 2 provides the mode bias fluents for the sentences in a story.

Reasoning. The reasoning module attempts to answer a question using the information extracted from the story and the learned hypothesis. It involves automatically generating and solving an ASP program that combines the ASP representations and learned hypothesis. Ideally, the correct solution to this program (i.e., the answer sets) will contain the correct answer to the question. To extract the answer from the reasoning output, we divide the questions into two types: *"yes/no/maybe"* and others. In the case of the former, we use a *representation search* that checks the question's representation against the answer sets based on the following criteria: (i) if there is at least one answer set, and the representation is in all answer sets, then return *"yes"*; (ii) if there is at least one answer set, and the representation is in some, but not all answer sets, then return *"maybe"*; (iii) otherwise the answer is *"no"*. For all other questions, we extract the answer using a *unification search*, i.e., by finding all ground atoms in the set of answer sets that unify with the question's formal representation. To identify these unifications, a regular expression is constructed from the question's formal representation, replacing variables with the wildcard expression *".*"*. Once unifications are detected, the ground terms corresponding to the *".*"* sections of the regular expressions are added to the answer list. For example, the question in Table 2 generates "$be\_afraid\_of\backslash(mary, .*\backslash)$". In case of a wrong predicted answer, the learner is invoked with the question and correct answer to learn from.

LAS Learning. Learning commonsense knowledge from story-based Q&A is initiated through the LAS Learning module. It takes the EC representations of the story and the question, generated by the ASP representation module and the correct and incorrect answers for the question, as input. It creates the context dependent learning task for ILASP by automatically generating the mode bias

Table 2. Sentences, fluent representations, and mode bias fluents for a short story.

Sentence	Fluent Representation	Mode bias fluents
Mary is a mouse.	`be(mary, mouse)`	`be(var(nnp), var(nn))`
Mice are afraid of wolves.	`be_afraid_of(mouse, wolf)`	`be_afraid_of(var(nn), var(nn))`
What is Mary afraid of?	`be_afraid_of(mary, V1)`	`be_afraid_of(var(nnp), var(nn))`

declarations, using the mode bias fluent representations, and the set E of CDPI examples. To create the mode bias declarations, the system checks whether the sentence is a question, or whether the base of its fluent predicate is "be". This two-check scheme suffices to generate the language bias, given our basic sentences and limited bAbI dataset vocabulary. For questions, the system aims to learn the concept introduced in it. So, its mode bias fluent representation becomes the argument of a mode head declaration, denoted in ILASP as *"modeh"*. If the sentence is not a question and the base of its fluent representation is "be", then ILASP *"modeb"* declarations are generated with the sentence's fluent as argument of mode body declaration. For the story presented in Table 2 the mode bias are:

```
#modeb(be(var(nnp),var(nn))).
#modeb(be_afraid_of(var(nn),var(nn))).
#modeh(be_afraid_of(var(nnp),var(nn))).
```

When LLM2LAS detects that the task requires reasoning about events, the language bias, referred to as EC mode bias, includes dedicated predicates. To learn the concept introduced by the question, LLM2LAS learns if it is *initiated* or *terminated*, considering also the initiation or termination of other fluents in the story. For the question's fluent, two mode head predicates are generated: *"initiatedAt"* and *"terminatedAt"* and declared as arguments of ILASP *"modeh"*. A body predicate is created by enclosing the question's fluent in a *"holdsAt"* predicate, which is then wrapped in *"modeb"*. For sentences that are not questions, the system detects if they describe an initiated state or a state that holds. In the first case a mode body predicate is created by enclosing the sentence's fluent into a *"initiatedAt"* predicate, in the second case the sentence's fluent is enclosed into a *"holdsAt"* predicate. Both become arguments of ILAPS *"modeb"* declarations. The EC mode bias declarations for the example in Table 2 are as follows:

```
#modeb(initiatedAt(be(var(nnp),var(nn)),var(time))).
#modeb(initiatedAt(be_afraid_of(var(nn),var(nn)),var(time))).
#modeb(holdsAt(be(var(nnp),var(nn)),var(time))).
#modeb(holdsAt(be_afraid_of(var(nn),var(nn)),var(time))).
#modeh(initiatedAt(be_afraid_of(var(nnp),var(nn)),var(time))).
#modeh(terminatedAt(be_afraid_of(var(nnp),var(nn)),var(time))).
#modeb(holdsAt(be_afraid_of(var(nnp),var(nn)),var(time))).
```

The formal representation of non-question sentences is used to prove or disprove other facts using the domain-independent EC rules in (1) and the learned hypothesis. So sentences that do not have a "be"-based verb, denote actions at the specific time point in the story. So their mode bias fluent representation becomes argument of the *"happensAt"* predicate. Consider the sentence *"Mary goes to the store."*, whose mode bias fluent is $\texttt{go\_to}(\texttt{var}(\texttt{nnp}), \texttt{var}(\texttt{nn}))$. The system generates the mode body declaration:

```
#modeb(happensAt(go_to(var(nnp),var(nn)),var(time))).
```

Another key step is the automatic generation of CDPI examples. Examples are created from questions, their stories, and their correct and incorrect answers.

Each example corresponds to an incorrectly answered question by the reasoning module, as this would trigger the learning module. Intuitively, the representations of all sentences prior to the question would form the example's context, the formal representations of the correct answer would be part the example's inclusion set and the formal representations of some of the incorrect answers would be part of the example's exclusion set. However, the formal presentation of the story may include choice rules, hence lead to multiple answer sets. When no choice rule is present, if the example is generated for a yes/no/maybe question and the correct answer is "yes" (resp. "no"), the representation of the question forms the CDPI example's inclusion (resp. exclusion) set and the exclusion (resp. inclusion) set is empty. Similarly for questions which are not yes/no/maybe questions. If choice rules are present in the formal representation of a story, the example generation has to take into account brave and cautious entailment. For a yes/no/maybe question, if the answer is "maybe", then the example will include the question representation in its inclusion set to guarantee that the question's concept occurs in at least one answer set. If the correct answer is a "yes", then a negative example is created where the inclusion set is empty and the exclusion set includes the question's correct answers. This is to guarantee that the question's formal representation is true in all answer sets. In all other cases, a negative example is created with an empty exclusion set and inclusion set given by the representation of the question's answers. This is to guarantee the wrong answer will be false in all answer sets.

To illustrate some of the cases explained above, consider the following story: *Daniel went to the kitchen. Daniel went to the bedroom.* The question: *Where is Daniel?* The correct answer is the *bedroom* and incorrect answer is *kitchen.* If the incorrect answer is predicted, then the following example is created:

```
%Background Knowledge
holdsAt(F,T+1) :- initiatedAt(F,T),time(T).
holdsAt(F,T+1) :- holdsAt(F,T), not terminatedAt(F,T),time(T).
%Mode bias
#modeb(happensAt(go_to(var(nnp),var(nn)),var(time))).
#modeb(holdsAt(be(var(nnp),var(nn)),var(time))).
#modeh(initiatedAt(be(var(nnp),var(nn)),var(time))).
#modeh(terminatedAt(be(var(nnp),var(nn)),var(time))).
%Positive and Negative Examples
#pos({holdsAt(be(daniel,bedroom),3)},{holdsAt(be(daniel,kitchen)
,3)},{time(1..3). happensAt(go_to(daniel,kitchen),1).
happensAt(go_to(daniel,bedroom),2).}).
```

The system requires minimal background knowledge. If Event Calculus (EC) is needed, the background knowledge includes only the rules in (1). When ILASP system is run to solve the generated learning task, if the task is satisfiable, the reasoner is updated with the learned hypothesis.

5 Empirical Evaluation

In this section, we initiate our evaluation of the proposed approach using the bAbI dataset from Facebook Research [20]. We first describe the dataset, and then provide a description of the hardware and software configurations we have employed and of the our baselines. Finally, we comment on the results that confirm the efficacy of our system.

Dataset. The bAbI dataset is a dataset comprising 20 non trivial tasks of text understanding and reasoning that was proposed by Facebook Research. The bAbI dataset was conceived as a benchmark for assessing a range of natural language reasoning abilities, including deduction, path finding, spatial reasoning, and counting [20]. Each task of the dataset is constructed by simulating words that represent entities and actions. An entity, denoted as a noun, can be a location, an object, or a person, and possesses internal attributes such as size, color, or relative position to cardinal directions. Within this simulation, each entity can perform ten fundamental actions, with each action associated with a collection of replacement synonyms, pronouns, and temporal adverbs, ensuring lexical diversity within the tasks. More in detail, the dataset comprises 4 actors, 6 locations, and 3 objects per task, it features stories ranging from 3 to 229 sentences and 1 to 12 questions. Each sentence within a story is uniquely identified and accompanied by its answer for each task. The dataset includes both training and test data for each task, with a strong focus on learning from a few examples. The stories are also available in human-readable formats in various languages. In this experiment, we place our focus on the natural language, examining the 13 tasks for which our implementation can be directly applied, namely tasks identified as: 1, 6, 8, 9, 10 11, 12, 13, 14, 15, 16, 18, and 20.

Hardware and Software Setup. All experiments are conducted on a machine equipped with Intel(R) Core(TM) i7-1255U processors, 16.0 GB of RAM, and GPU NVIDIA RTX 6000 Ada Generation with 48 GB of memory. The experiment pipelines are implemented in the Python programming language version 3.9. Our architecture has been implemented using spaCy, a free open-source library for Natural Language Processing, for basic natural language processing tasks (e.g., POS tagging), the open-source LLM Falcon, Clingo 5.6.2 for reasoning on ASP programs, and ILASP 4.0 for learning from answer sets. Concerning the learning parameters, the maximum penalty for the size of the hypothesis was set to 50, and the maximum number of variables was set to 3 for the tasks solved with fluent representation, and to 4 for the tasks solved with EC representation. Each task has been evaluated on 1000 training examples. Our implementation has been compared against two baselines from the literature, also based on logic programming: the ILP-based system in Mirta et al. [13], and the approach proposed by Yang et al. [21]. All the material needed to reproduce our experiments can be downloaded from https://github.com/IrfanKareem/llm2las/tree/LPNMR24.

Results and Discussion. Our experimental results are summarized in Table 3, that provides the accuracy of each task. The task 14, 16, 18, and 20 were solved

Table 3. Performance of compared methods in terms of accuracy.

Task	LLM2LAS	Yang et al. [21]	Mirta et al. [13]
1 Single supporting fact	100	100	100
6 Yes/no questions	100	100	100
8 Lists/sets	100	100	100
9 Simple negation	100	100	100
10 Indefinite knowledge	100	100	100
11 Basic coreference	100	100	100
12 Conjunction	100	100	100
13 Compound coreference	100	100	100
14 Time reasoning	100	100	100
15 Basic deduction	100	100	100
16 Basic induction	100	100	93.6
18 Size reasoning	100	100	100
20 Agents motivations	100	100	100

with fluent representation. Task 14 tests the system's ability to understand the use of time expressions; Task 18 assesses the system's capability to reason about the relative sizes of objects; Task 20 reasons about agent motivations, encompassing the state of an agent (e.g., tired, hungry) and actions such as moving to a location or picking up an object. The remaining tasks (1, 6, 8, 9, 10, 11, 12, 13, 15) are solved with EC representation. These tasks assess the system's ability to reason about object and agent locations, discern true or false statements from mixed information, generate single-word answers based on object properties, identify negations, handle ambiguous queries, detect coreference and conjunction, and deduce basic agents (i.e. animal) properties.

The learning times for our system has an average (over all tasks) of 58 s, with a peak of 210 s for the hardest task 16, and a minimum of 2.8 s for task 14. Concerning the accuracy of answers provided, we observe that our system achieves 100% accuracy on all tasks. Interestingly, the encoding learnt are substantially the same as the ones provided by human experts in Yang et al.'s [21]. In conclusion, our system successfully learns and reason and provide answers over 13 commonsense driven tasks in the bAbI dataset, matching the performance of human-expert manually engineered ASP programs.

6 Conclusion and Future Work

In this paper, we propose to combine LLM with learning from answer sets to correctly solve story-based Q&A tasks. Specifically, our system combines the LLM Falcon with LAS for learning commonsense knowledge and ASP for reasoning over given stories in order to correctly answer questions by means of the learned knowledge. The missing commonsense knowledge is learnt from examples of wrong predicted answers using very minimal background knowledge. Our system shows very promising performance on the bAbI dataset. Our approach can be easily extended to other (more involved) datasets by using larger LLMs with higher text understanding and Semantic Parsing capabilities. These improvements are intended for future work. In the future, we plan to solve the remaining bAbI tasks and make the system more general purpose by extending the tasks done by a LLMs. We also aim to scale up by using FastLAS for non-recursive learning, and explore the use of LLMs also for data processing.

Acknowledgments. This work was partially supported by the Italian Ministries MIMIT, under project EI-TWIN n. F/310168/05/X56 CUP B29J24000680005, and MUR, under projects: PNRR FAIR - Spoke 9 - WP 9.1 CUP H23C22000860006, Tech4You CUP H23C22000370006, and PRIN PINPOINT CUP H23C22000280006.

References

1. Al-Negheimish, H., Madhyastha, P., Russo, A.: Numerical reasoning in machine reading comprehension tasks: are we there yet? In: EMNLP (1), pp. 9643–9649. Association for Computational Linguistics (2021)
2. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
3. Calimeri, F., et al.: ASP-Core-2 input language format. Theory Pract. Log. Program. **20**(2), 294–309 (2020)
4. Davis, E., Marcus, G.: Commonsense reasoning and commonsense knowledge in artificial intelligence. Commun. ACM **58**(9), 92–103 (2015)
5. Drozdov, A., et al.: Compositional semantic parsing with large language models. In: ICLR. OpenReview.net (2023)
6. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, vol. 88, pp. 1070–1080 (1988)
7. Ishay, A., Yang, Z., Lee, J.: Leveraging large language models to generate answer set programs. In: KR, pp. 374–383 (2023)
8. Jurafsky, D., Martin, J.H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2nd edn. Prentice Hall, Pearson Education International (2009)
9. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Gener. Comput. **4**, 67–95 (1986)
10. Lake, B.M., Murphy, G.L.: Word meaning in minds and machines. CoRR abs/2008.01766 (2020)
11. Law, M.: Inductive learning of answer set programs. Ph.D. thesis, London, UK (2018)
12. Lifschitz, V.: What is answer set programming? In: AAAI, pp. 1594–1597. AAAI Press (2008)
13. Mitra, A., Baral, C.: Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In: AAAI, pp. 2779–2785. AAAI Press (2016)
14. Nye, M.I., Tessler, M.H., Tenenbaum, J.B., Lake, B.M.: Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. In: NeurIPS, pp. 25192–25204 (2021)

15. Perzanowski, A.: Memory and reasoning in deep learning: data efficiency of the SAM-based two-memory (STM) model (2022)
16. Ray, O.: Nonmonotonic abductive inductive learning. J. Appl. Log. **7**(3), 329–340 (2009)
17. Sap, M., Shwartz, V., Bosselut, A., Choi, Y., Roth, D.: Commonsense reasoning for natural language processing. In: ACL (tutorial), pp. 27–33. ACL (2020)
18. Vaswani, A., et al.: Attention is all you need. In: NIPS, pp. 5998–6008 (2017)
19. Wei, J., et al.: Chain-of-thought prompting elicits reasoning in large language models. In: NeurIPS (2022)
20. Weston, J., Bordes, A., Chopra, S., Mikolov, T.: Towards AI-complete question answering: a set of prerequisite toy tasks. In: ICLR (Poster) (2016)
21. Yang, Z., Ishay, A., Lee, J.: Coupling large language models with logic programming for robust and general reasoning from text. In: ACL (Findings), pp. 5186–5219. Association for Computational Linguistics (2023)
22. Zheng, S., Huang, J., Chang, K.C.C.: Why does ChatGPT fall short in answering questions faithfully? arXiv preprint arXiv:2304.10513 (2023)