Declarative Question Answering over Knowledge Bases containing Natural Language Text with Answer Set Programming

Arindam Mitra[‡] and **Peter Clark**[†] and **Oyvind Tafjord**[†] and **Chitta Baral**[‡]

[†] Allen Institute for Artificial Intelligence, Seattle, WA, U.S.A.
[‡] Arizona State University, AZ, U.S.A.
{peterc,oyvindt}@allenai.org,{amitra7,chitta}@asu.edu

Abstract

While in recent years machine learning (ML) based approaches have been the popular approach in developing end-to-end question answering systems, such systems often struggle when additional knowledge is needed to correctly answer the questions. Proposed alternatives involve translating the question and the natural language text to a logical representation and then use logical reasoning. However, this alternative falters when the size of the text gets bigger. To address this we propose an approach that does logical reasoning over premises written in natural language text. The proposed method uses recent features of Answer Set Programming (ASP) to call external NLP modules (which may be based on ML) which perform simple textual entailment. To test our approach we develop a corpus based on the life cycle questions and showed that Our system achieves up to 18% performance gain when compared to standard MCQ solvers.

Developing intelligent agents that can understand natural language, reason and use commonsense knowledge has been one of the long term goals of AI. To track the progress towards this goal, several question answering challenges have been proposed (Levesque, Davis, and Morgenstern 2012; Clark et al. 2018; Richardson, Burges, and Renshaw 2013; Rajpurkar et al. 2016). Our work here is related to the school level science question answering challenge, ARISTO (Clark 2015; Clark et al. 2018). As shown in (Clark et al. 2018) existing IR based and end-to-end machine learning systems work well on a subset of science questions but there exists a significant amount of questions that appears to be hard for existing solvers. In this work we focus on one particular genre of such questions, namely questions about life cycles (and more generally, sequences), even though they have a small presence in the corpus.

To get a better understanding of the "life cycle" questions and the "hard" ones among them consider the questions from Table 1. The text in Table 1, which describes the life cycle of a frog does not contain all the knowledge that is necessary to answer the questions. In fact, all the questions require some additional knowledge that is not given in the text. Question 1 requires knowing the definition of "middle" of a sequence. Question 2 requires the knowledge of "between". Question 3 on other hand requires the knowledge of "a good indicator". Note that for question 3, knowing whether an adult frog has lungs or if it is the adult stage where the frog loses its tail is not sufficient to decide if option (A) is the indicator or option (B). In fact an adult frog satisfies both the conditions. An adult frog has lungs and the tail gets absorbed in the adult stage. It is the uniqueness property that decides that option (B) is an indicator for the adult stage. We believe to answer these questions the system requires access to this knowledge.

Since this additional knowledge of "middle", "between", "indicator" (and some related ones which are shown later)

Life Cycle of a Frog
order: egg → tadpole → tadpole with legs → adult
egg - Tiny frog eggs are laid in masses in the water by a female frog. The eggs hatch into tadpoles.
tadpole - (also called the polliwog) This stage hatches from the egg. The tadpole spends its time swimming in the water, eating and growing. Tadpoles breathe using gills and have a tail.
tadpole with legs - In this stage the tadpole sprouts legs (and then arms), has a longer body, and has a more distinct head. It still breathes using gills and has a tail.
froglet - In this stage, the almost mature frog breathes with lungs and still has some of its tail.
adult - The adult frog breathes with lungs and has no tail (it has been absorbed by the body).
1. *What is the middle stage in a frogs life? (A) tadpole with legs (B) froglet*
2. *What is a stage that comes between tadpole and adult in the life cycle of a frog? (A) egg (B) froglet*
3. *What best indicates that a frog has reached the adult stage? (A) When it has lungs (B) When its tail has been absorbed by the body*

Table 1: A text for life cycle of a Frog with few questions.

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

is applicable to any sequence in general and is not specific to only life cycles, we aim to provide this knowledge to the question answering system and then plan to train it so that it can recognize the question types. The paradigm of declarative programming provides a natural solution for adding background knowledge. Also the existing semantic parsers perform well on recognizing questions categories. However the existing declarative programming based question answering methods demand the premises (here the life cycle text) to be given in a logical form. For the domain of life cycle question answering this seems a very demanding and impractical requirement due to the wide variety of sentences that can be present in a life cycle text. Also a life cycle text in our dataset contains 25 lines on average which makes the translation more challenging.

The question that we then address is, "can the system utilize the additional knowledge (for e.g. the knowledge of an "indicator") without requiring the entire text to be given in a formal language?" We show that by using Answer Set Programming and some of its recent features (function symbols) to call external modules that are trained to do simple textual entailment, it is possible do declaratively reasoning over text. We have developed a system following this approach that answers questions from life cycle text by declaratively reasoning about concepts such as "middle", "between", "indicator" over premises given in natural language text. To evaluate our method a new dataset has been created with the help of Amazon Mechanical Turk. The entire dataset contains 5811 questions that are created from 41 life cycle texts. A part of this dataset is used for testing. Our system achieved up to 18% performance improvements when compared to standard baselines.

Our contributions in this work are two-fold: (a) we propose a novel declarative programming method that accepts natural language texts as premises, which as a result extends the range of applications where declarative programming can be applied and also brings down the development time significantly; (b) we create a new dataset of life cycle texts and questions, which contains annotated logical forms for each question.

Background

Answer Set Programming

An Answer Set Program is a collection of rules of the form,

$$L_0 \text{ :- } L_1, ..., L_m, \textbf{not } L_{m+1}, ..., \textbf{not } L_n.$$

where each of the L_i's is a literal in the sense of classical logic. Intuitively, the above rule means that if $L_1, ..., L_m$ are true and if $L_{m+1}, ..., L_n$ can be safely assumed to be false then L_0 must be true (Gelfond and Lifschitz 1988). The left-hand side of an ASP rule is called the *head* and the right-hand side is called the *body*. The symbol :- ("if") is dropped if the *body* is empty; such rules are called facts. Throughout this paper, predicates and constants in a rule start with a lower case letter, while variables start with a capital letter. The following ASP program represents question 3 from Table 1 with three facts and one rule.

Listing 1: a sample question representation

```
qIndicator(frog,adult).
option(a, has(lungs)).
option(b, hasNo(tail)).
ans(X):- option(X,V), indicator(O,S,V),
         qIndicator(O,S).
```

The first *fact* represents that question 3 is an 'indicator' type question and is looking for something which indicates that a *frog* is in the *adult* stage. The later two *facts* roughly describes the two answer choices, namely "(a) when it has lungs" and "(b) when its tail has been absorbed by the body". The last rule describes that for an indicator type question, the option number X is a correct answer if the answer choice V is an indicator for the organism O being in stage S i.e. if $indicator(O, S, V)$ is true.

Aggregates A rule in ASP can contain aggregate functions. An aggregate function takes as input a set. ASP has four built-in aggregates namely *#count, #max, #min, #sum* which respectively computes the number of elements in a set, the maximum, minimum or the sum of numbers in the set. The follows rule defines the concept of an 'indicator' using the *#count* aggregate.

Listing 2: Defining Indicator of a stage

```
indicator(O,Stage,P) :-
   stageFact(O,Stage,P),
   #count {stageFact(O,S1,P)} = 1.
```

Here, $stageFact(O, Stage, P)$ captures the attributes P that are true when the organism O is in the stage S. The above rule then describes that P is an indicator for O being in stage S if P is true in S and it is only true in S i.e. the total number of stages $S1$ where $Prop$ is true is one.

String valued Terms The object constants in ASP can take *string* values (written within quotes " "). This is useful while working with text. For example, the options in the question 3 can also be represented as follows:

```
option(a, "when it has lungs").
option(b, "when its tail has
           been absorbed by the body").
```

Function Symbols A *function symbol* allows calling an external function which is defined in a scripting language such as *lua* or *python* (Calimeri et al. 2008). An occurrence of a function symbol making an external call is preceded by the '@' symbol. For e.g., $@stageFact(O, S, P)$ denotes a function symbol that calls to an external function named $stageFact$ which takes three arguments as input. A function symbol can return any simple term such as *name*, *number* and *strings* as output.

QA using Declarative Programming

A question answering (QA) system that follows declarative programming approach primarily requires three components: a **semantic parser** $\mathcal{SP}$, a **knowledge base** $\mathcal{KB}$ and a **set of rules** (let's call it **theory**) $\mathcal{T}$.

- The goal of the **semantic parser** $\mathcal{SP}$ is to translate a given question into a logical form.
- The $\mathcal{KB}$ provides facts or "premises" with respect to which the question should be answered. For e.g. for the frog life cycle the $\mathcal{KB}$ might look like the following:

Listing 3: A sample KB for part of the Frog life cycle

```
stageFact(frog,tadpole,has(tail)).
stageFact(frog,froglet,has(lungs)).
stageFact(frog,froglet,has(tail)).
stageFact(frog,adult,has(lungs)).
stageFact(frog,adult,hasNo(tail)).
```

- The *theory* $\mathcal{T}$ contains inference enabling rules.

To answer a question, the system first translates the question into a logical form and then combines that with the $\mathcal{KB}$ and the $\mathcal{T}$ to create a consolidated program. The output (models) of which provides the answer.

For the running example of the 'indicator' question ($Q3$ from Table 1) if the theory $\mathcal{T}$ contains the rule in listing 2, some semantic parser provides the question representation in listing 1 and the $\mathcal{KB}$ contains the facts in listing 3, then the output will contain the deduced fact $ans(b)$ describing that option (B) is the correct answer. This is because, the rule in listing 2 will deduce from the $\mathcal{KB}$ that $indicator(frog, adult, hasNo(tail))$ is true. The last rule in listing 1 will then conclude that $ans(b)$ is true. Since there is no reason to believe that $indicator(frog, adult, has(lungs))$ is true, $ans(a)$ will not be part of the output (model). The semantics of ASP is based on the stable model semantics (Gelfond and Lifschitz 1988). For further details interested readers can refer to (Gebser et al. 2012; Gelfond and Kahl 2014).

Proposed Approach

The issue in the running example is that it is difficult to get the facts in terms of $stageFact/3$ predicate and in the actual $\mathcal{KB}$ we do not have facts in this format. Rather we have the life cycle texts (Table 1) describing the facts. To deal with this we replace such predicates with two external function symbols, namely *generate* and *validate*.

Generate A *generate function* for a predicate takes the arguments of the predicate and returns a textual description of the predicate instance following some template. For example, a generate function for *stageFact* can take *(frog, adult, hasNo(tail))* as input and returns a string such as "an adult frog has no tail" or if it for predicate named *parent* it can take *(x, y)* and return "x is a parent of y".

Validate A *validate function* takes a string describing a proposition (e.g. "an adult frog has no tail") and validates the truthfulness of the proposition against a $\mathcal{KB}$ containing text (e.g. Table 1). For now let us assume a *validate function* returns 1 or 0 depending on whether the proposition is true or false according to the text in the $\mathcal{KB}$.

With this transformation the "indicator" rule from listing 2 will look as follows:

```
indicator(O,Stage,Prop) :-
  P = @g_StageFact(O,Stage,Prop),
  @v_StageFact(P) ==1,
  #count { S1: v_StageFact(P1)==1,
    P1 = @g_StageFact(O,S1,Prop)} == 1.
```

The above rule could be read as follows: $Prop$ denotes that O is in stage S if the natural language description of $StageFact(O, Stage, Prop)$ which is obtained by calling the $g\_StageFact$ function is true according to the $v\_StageFact$ function and also the number of stages $S1$ where the natural language description of $StageFact(O, S1, Prop)$ true is equal to 1.

The pair of **generate-validate** function symbols delegates the responsibility of verifying if a proposition is true or not to an external function and we believe that if the proposition is simple enough and close to the texts described in a KB, a simple **validate** function might be able to compute the truth value with good accuracy. However, one important issue with this rule is that it is not "safe". In simple words the above rule does not specify what values the variables $O, Stage, Prop, S1$ can take and as a result what to pass as arguments to the $g\_StageFact$ functions is undefined. To mitigate this issue one needs to add some domain predicates which describes the possible values of the unbounded variables. For our question answering task, we have used the predicates that represent the question as domain predicates. The resulting rule, then, looks as follows:

```
indicator(O,Stage,Prop) :-
  qIndicator(O, Stage),
  qOption(X, Prop),
  P = @g_StageFact(O,Stage,Prop),
  @v_StageFact(P) ==1,
  #count { S1: isAStageOf(S1, O),
    v_StageFact(P1)==1,
    P1 = @g_StageFact(O,S1,Prop)} == 1.
```

The $isAStageOf(S1, O)$ describes the stages in the life cycle of the organism O and is extracted from the "order" field in life cycle texts ("Order" in Table 1).

On the choices of a Validate function The task of deciding if a proposition is true based on a given text is a much studied problem in the field of NLP and is known as *textual entailment*. There exist several textual entailment functions. All of which can be used as validate function. However, the textual entailment functions returns a real value between 0 to 1 denoting the probability that the proposition is true and thus one needs to decide a threshold value to obtain a boolean validate function. In the implementation of our system we have not used a boolean validate function but used the entailment score as it is. We describe how to use a fuzzy validate function in the next section after describing the life cycle dataset and the representation of the texts.

The Dataset and The Implemented System

The life cycle question answering dataset contains a total of 41 texts and $5.8k$ questions. Each text contains a *sequence* which describes the order of stages and a *natural language description* of the life cycle as shown in Table 1. The life cycle texts are collected from the internet. The sequence of the stages are manually added by either looking at the associated image in the website that describes the order of the stages or from the headings of the text (Table 1).

Representing Life Cycle Texts Each life cycle text is represented in terms of two predicates, namely, *stageAt(URL, O, Pos, S)* and *description(URL, O, T)*. The $stageAt$ predicate describes that according to the source *URL* (from which the text is collected) the stage that comes at position P in the life cycle of O is S. The *description* stores the text that describes the life cycle. The following ASP program shows the representation of the text in Table 1. To save space the actual value of the *URL* is replaced by 'u'. The $\mathcal{KB}$ contains representations of 41 such texts.

```
stageAt(u,"frog",1,"egg").
stageAt(u,"frog",2, "tadpole").
stageAt(u,"frog",3,"tadpole with legs")
stageAt(u,"frog",4,"froglet").
stageAt(u,"frog",5,"adult").
description(u,"frog",
     "Egg: Tiny frog eggs are laid...").
```

Question categories

The question that are created from these texts are divided into 11 categories. The first three types of questions namely *look up, difference, indicator* require reading the textual description of stages whereas the remaining six types of questions can be answered solely from the sequence of stages (egg $\rightarrow$ tadpole $\rightarrow$ tadpole with legs $\rightarrow$ adult).

Look Up Questions This category contains questions the answer to which can be directly looked up from the description of the stages and does not require any special thinking. The following list shows some questions in this category:

How do froglets breath? (A) using lungs (B) using gills

The tail of a frog disappears at what stage? (A) adult (B) froglet

Where do female frogs lay their eggs? (A) In water (B) On land

Difference Questions This category of questions compare two stages based on their physical attributes, abilities or need that is true in one stage but not in other. The following list shows examples:

What is an adult newt able to do that a tadpole cannot? (A) walk on land (B) swim in water

A tadpole just turned into an eft. What does it need now? (A) shade (B)water

A seedling develops what that a sprout does not have? (A) protective bark (B) root

Indicator Questions This category of questions mentions an organism, a stage, two answer choices and asks which one of those indicates that the organism is in the given stage. Question 3 in Table 1 provides an example of this.

Sequence Based Questions Questions from this category can be answered based on the sequence of stages that describes journey of an organism from beginning to the end (e.g. egg $\rightarrow$ tadpole $\rightarrow$ tadpole with legs $\rightarrow$ adult). Questions in this category are further divided into 8 classes which takes one of the following forms:

Next Stage Questions Given a stage and an organism, asks for the next stage.

Before Stage Questions Given a stage and an organism, asks for the stages that appear before.

Between Stage Questions Given two stages and an organism, asks for the stages that appear between those two.

Stage At Questions Given an organism and a position, asks for the stages that appear at that position.

Count Questions Given an organism asks how many stages are there in the life cycle.

Correctly Ordered Questions Given an organism asks the sequence that describes the correct order of the stages.

Stage Of Questions Given an organism asks for the stages that appear in its life cycle.

Not a Stage of Questions Given an organism asks for the stages that do not appear in its life cycle.

Table 2 shows an example of each types of questions.

Question Representation The representation of a question comprises of four ASP facts. Given an MCQ question of the form "$\langle$Q?$\rangle$ (A) $\langle$answer choice 1$\rangle$ (B) $\langle$answer choice 2$\rangle$ ", the first three facts are computed trivially as follows:

```
question(``Q?'').
option(a,``answer choice 1'').
option(b,``answer choice 2'').
```

The fourth fact captures the type of the question (i.e. look up, difference etc.) and some associated attributes (i.e. organism, stages, position). For each one of the 11 types of questions in the dataset there is a fixed template which describes the associated attributes for each type of question. The fourth fact is an instantiation of that template which is computed by a semantic parser. Table 2 describes the questions templates and shows an example instantiation.

Theory

The *theory* contains a total of 36 ASP rules, 3 *generate* functions one for each of the *look up, difference* and *stage indicator* question type and a single *validate* function. The *validate* function, $@validate(Text, Hypothesis)$ takes as input a life cycle *text* and a *hypothesis* (string) and returns a score between 0 to 1. The score is computed using a textual entailment function as follows:

$$score = max\{\ textual\_entailment(S, Hypothesis) : S\ \textit{is a sentence in}\ Text\}$$

To find the answer a confidence score $V \in [0, 1]$ is computed for each answer option X (denoted by $confidence(X, V)$). The rules in the *theory* computes these confidence scores. The correct answer is the option that gets maximum confidence score. The following rule describes this:

```
ans(X):- option(X,V), confidence(X,V),
       V == #max {V1:confidence(X1,V1)}.
```

Due to limited space we only describe the rules that call entailment functions through function symbols.

question template	example question	instantiated template	#Questions
$qLookup(O)$	How do froglets breath?	$qLookup(\text{“}frog\text{”})$	2525
$qDifference(O, S1, S2)$	What is an adult newt able to do that a tadpole cannot?	$qDifference(\text{“}newt\text{”}, \text{“}tadpole\text{”}, \text{“}adult\text{”})$	167
$qIndicator(O, S)$	When do you consider a penguin to have reached the adult stage?	$qIndicator(\text{“}penguin\text{”}, \text{“}adult\text{”})$	125
$qNextStage(O, S)$	A salmon spends time as which of these after emerging from an egg?	$qNextStage(\text{“}salmon\text{”}, \text{“}egg\text{”})$	346
$qStageBefore(O, S)$	Newt has grown enough but it is not yet in the tadpole stage, where it might be?	$qStageBefore(\text{“}newt\text{”}, \text{“}tadpole\text{”})$	123
$qStageBetween(O, S1, S2)$	What is the stage that comes after egg and before eft in the newt life cycle?	$qStageBetween(\text{“}newt\text{”}, \text{“}egg\text{”}, \text{“}eft\text{”})$	123
$qStageAt(O, P)$	What stage a longleaf pine will be in when it is halfway through its life?	$qStageAt(\text{“}longleaf\ pine\text{”}, middle)$	520
$qCorrectlyOrdered(O)$	To grow into an adult, fleas go through several stages. Which of these is ordered correctly?	$qCorrectlyOrdered(\text{“}flea\text{”})$	43
$qCountStages(O)$	From start to finish, the growth process of a wolf consists of how many steps?	$qCountStages(\text{“}wolf''')$	113
$qIsAStageOf(O)$	The growth process of lizards includes which of these?	$qIsAStageOf(\text{“}lizard\text{”})$	1500
$qIsNotAStageOf(O)$	To grow into an adult, fleas go through 4 stages. Which is not one of them?	$qIsNotAStageOf(\text{“}flea\text{”})$	227

Table 2: Question templates and total number of questions for each question category. Variables starting with O, P, S respectively refers to an organism, a position and a stage. A position could be a natural number or any member of {middle, last}.

Lookup Questions Given the representation of a lookup question such as: {*qLookup(“frog”). question(“How do froglets breathe?”). option(a,“using gills”). option(b,“using lungs”).*}, the following rule computes the confidence score
for each option.

```
confidence(X,V):-
  question(Q), qOption(X,C),
  H = @generate_lookup(Q,C),
  qLookup(Org), description(URL,Org,P),
  V = @validate(P,H),
```

While creating the confidence for option “a” this rule will call the *generate_lookup(Q,C)* function with Q = “How do froglets breathe?” and C = “using gills”. The *generate_lookup* function then returns a hypothesis “froglets breathe using gills”. The *validate* function then takes the description of the frog life cycle and the hypothesis and verifies if any of the sentence in the text supports the hypothesis: “froglets breathe using gills”. The confidence score of option “a” is the score returned by the *validate* function. Similarly it will compute the confidence score for option “b”.

The work of (Khot, Sabharwal, and Clark 2018) presents a function that creates a hypothesis from a question and an answer choice which was used to solve MCQ questions. The *generate_lookup* function here reuses their implementation.

Difference Questions Given a difference question (e.g. “What is an adult newt able to do that a tadpole cannot?” and an answer choice (e.g. “walk on land”) a *generate* function returns two hypothesis H_1 and H_2. (“adult newt able to walk on land”, “a tadpole cannot walk on land”). The fuzzy truth value for each each hypothesis is computed with the $validate$ function. The product of which is assigned to be the confidence score of the answer choice. A rule is written in ASP to describe the same.

Indicator Questions When dealing with a fuzzy *validate* function the definition of an *indicator* is modified as follows: Let v be the score that an answer choice c indicates that the organism O is in stage S. If O goes through n stages, S represents the j-th stage and p_i is the truth value that c is true in stage i, then $v = p_j * \prod_{k=1,k\neq j}^{n}(1 - p_k)$. The following five ASP rules are written to describe the same.

```
stageIndicatorIndex(ID):-
   stageAt(URL, O, ID, S),
   qStageIndicator(O,S).

trueForStage(Idx,X,V):-qIndicator(O,S),
  option(X,C),stageAt(URL,O, Idx, S1),
  H = @generate_indicator(S1,C)
  description(URL, O, Text),
  V = @validate(Text, H).
```

```
result(1, X , @product("1.0",V,1,ID)):-
  trueForStage(O, 1,X,V),
  stageIndicatorIndex(SRC,ID).

result(O, N, X, @product(V1,V2,N,ID)):-
   res(O, N-1, X , V1),
   trueForStage(O, N,X,V2),
   stageIndicatorIndex(ID).

confidence(X,V):- res(N, X , V),
  N = #max {P:stageAt(URL,O, P, S )}.
```

The first rule finds out the index of the stage specified in the question. The second rule computes the truth value p_i (*trueForStage(Idx, X, V)*) for each stage index *Idx* and each option X. The last three rules compute the confidence score $v = p_j * \prod_{k=1,k\neq j}^{n}(1-p_k)$ iteratively. Here *product(V1, V2, N, ID)* function returns either $V1 * V2$ or $V1 * (1-V2)$ depending on whether N is equal to ID. The *generate_indicator* function follows a simple template. It takes as input a stage such as "froglet" and an answer choice, for e.g. "when it has lungs" and returns "In the ⟨froglet⟩ stage, ⟨it has lungs⟩".

Dataset Creation

We crowdsourced the dataset of 5811 multiple-choice life cycle questions with their logical forms with the help of Amazon Mechanical Turk. The workers did not create the logical forms. We collected them using reverse-engineering without exposing the workers to the underlying formalism.

To obtain the sequence based questions we followed the crowdsourcing technique in (Wang, Berant, and Liang 2015). Using $stageAt$ predicates in the $\mathcal{KB}$ and the rules in the *theory* we first computed a database of sequence based facts such as $nextSatge(frog, egg, tadpole)$. We then used a simple grammar to create an MCQ question out of it, for e.g, "What stage comes after egg stage in frog's life? (A) tadpole (B) adult". Finally we asked the workers to rephrase these questions as much as possible. Since the seed questions were generated using logical facts we could also compute the logical form and the correct answer beforehand.

To collect indicator type questions we gave the workers a life cycle text and described what is meant by an stage indicator question. Each worker were then asked to create two multiple choice stage indicator questions and write down the correct option and associated stage for each question. There were two workers working on each text. As a result we got $41 \times 2 \times 2 = 164$ questions. We manually removed the questions that did not meet the requirements and finally ended up with 125 questions. Using the stage name that was written down for each question we were able to compute the logical form $qStageIndicator(organism, stage)$. Similarly, a separate task was created to collect *stage difference* questions where the workers apart from the question and the answer choices also wrote down the two stages that are being compared. Using that we computed the logical form.

To obtain look up questions we gave the workers a life cycle text and asked them to create free form MCQ questions, which gave us 2710 questions. We then manually filtered the questions that should belong to the other 10 categories and ended up with 2525 look up questions. Since the question template of a look up question only contains the organism name we did not need any extra supervision to create the logical form.

Related work

Many question answering systems (Sharma et al. 2015; Mitra and Baral 2016; 2015; Wang, Lee, and Kim 2017; Lierler, Inclezan, and Gelfond 2017; Clark, Dalvi, and Tandon 2018) have been developed that use declarative programming paradigm. Among these the closest to our work are the works of (Lierler, Inclezan, and Gelfond 2017; Mitra and Baral 2016; Clark, Dalvi, and Tandon 2018) which try to answer a question with respect to a given text. But to do so they convert the associated text into some action language with existing natural language parsers (Bos 2008; He et al. 2017; Flanigan et al. 2014). Having a formal representation of the text is helpful but the ability to provide special domain knowledge should not be impaired by the absence of a formal representation of the text. Our work can be considered as a step towards that direction.

Our work is also related to (Eiter et al. 2006; Havur et al. 2014). Eiter et al. have used function symbols (referred to as *external atoms*) to interface ASP with an ontology language (e.g. OWL) that has different formats and semantics. In (Havur et al. 2014) function symbols are used to delegate some low level feasibility checks (such as "is it possible to move left without colliding") in a robotics application.

The task of textual entailment (Dagan, Glickman, and Magnini 2006) and semantic parsing (Zelle and Mooney 1996) play a crucial role in our work. With access to new datasets both the task have received significant attention (Bowman et al. 2015; Parikh et al. 2016; Chen et al. 2018; Wang, Berant, and Liang 2015; Krishnamurthy, Dasigi, and Gardner 2017).

Finally, recently there has been a surge of new question answering datasets. Depending on their restrictions on the possible answers they can be divided into three categories: (1) the answer is an exact substring of the text (2) the answer can take values from a fixed which is decided by the training dataset and (3) multiple choice questions. We have used the accuracy of existing science MCQ solvers (Khot, Sabharwal, and Clark 2018) as baselines in our experiment.

Experiments

Setup To evaluate our system we divide the 41 texts and the 5811 questions in two different ways and report the accuracy on both:

Text Split : In this case, we follow the machine comprehension style question answering and divide the 41 life cycle texts into three sets. The training set then contains 29 texts and $4k$ associated questions, the dev set contains 4 texts and 487 questions and the test set contains 8 texts with 1368 questions. Given a text and a MCQ question the task is to find the correct answer choice.

Question Split : In this split we mimic the open book exam setting and divide the $5.8k$ questions randomly into train, dev and test set each containing 4011, 579 and 1221 questions respectively. Here the knowledge base contains all the texts. Given a MCQ question the task is to find out the correct answer choice with respect to the knowledge base.

Our System We experiment with four different textual entailment functions. One of those is a neural network based model (Parikh et al. 2016). The remaining three are variations of n-grams and lexical similarity based model (Jijkoun and De Rijke 2006). The first variation (NGRAM-LS-1) uses WordNet based lexical similarity. The second variation uses (NGRAM-LS-2) weighted words (Jijkoun and De Rijke 2006) along with simple synonym based similarity. The third variation (NGRAM-LS-3) uses both word weights and WordNet based lexical similarity.

The semantic parser in (Krishnamurthy, Dasigi, and Gardner 2017) is trained to obtain the question template instances (e.g. $qIndicator(\text{“}frog\text{”}, \text{“}adult\text{”})$). We observed that the semantic parser predicts the question types (e.g. $qIndicator$) with high accuracy but often make errors in identifying the associated attributes (e.g. “adult”). For example it predicts that a given question is of *qStageAt* type with 100% accuracy but fails to identify the associated stage index attribute 38% times. Since the question templates in our dataset is quite simple and only contains one organism name, maximally two stage names or one stage index, we employ a simple search to extract the attributes. The resulting semantic parser then works as follows: it first obtains the question type from the trained parser of (Krishnamurthy, Dasigi, and Gardner 2017). Then it calls a function with a list containing all the organism names and the question. The function then returns the specified organism based on the first organism name that appears in the question. Similarly it makes subsequent calls for extracting stage names and positions. From now on we refer to the semantic parser in (Krishnamurthy, Dasigi, and Gardner 2017) as “KDG” and the customized version as “Customized-KDG”.

Baselines We use the performance of the entailment functions as baseline scores. For each option a hypothesis is created by combining the question and the answer choice using the code from (Khot, Sabharwal, and Clark 2018), which is then passed to an entailment function to compute the confidence score. A second set of baseline is computed using BiDaF (Seo et al. 2016) which performed well across several machine comprehension tasks. Given a passage and a question, BiDaF returns a substring of the passage as an answer. We then use that substring to compute the confidence score for each option. Two versions of BiDaF is used: BiDaF-1 which is trained on (Rajpurkar et al. 2016) and BiDaF-2 which is trained on both (Rajpurkar et al. 2016; Clark et al. 2018). To make the comparison fair, we have added a sentence of the type “The i-th stage is S” for each $stageAt(O, I, S)$ fact in the $\mathcal{KB}$. Also during the evaluation of “Question Split” only the necessary life cycle text is given as the passage.

Results

Table 3 presents the performance of all the systems on both splits. The first four rows show the accuracy of our system when gold representation of the question is used. This shows the best performance that the system can achieve with

System	Accuracy(%) Question Split	Accuracy (%)Text Split
Gold + (Parikh et al. 2016)	73.63	78.87
Gold + NGRAM-LS-1	78.95	**84.06**
Gold + NGRAM-LS-2	79.20	83.77
Gold + NGRAM-LS-3	**79.28**	83.77
KDG + (Parikh et al. 2016)	70.60	72.51
KDG + NGRAM-LS-1	73.87	**76.17**
KDG + NGRAM-LS-3	74.28	75.88
KDG + NGRAM-LS-3	**74.61**	76.02
Custom-KDG + (Parikh et al. 2016)	72.40	76.68
Custom-KDG + NGRAM-LS-1	77.07	**80.70**
Custom-KDG + NGRAM-LS-2	77.72	80.41
Custom-KDG + NGRAM-LS-3	**77.80**	80.48
(Parikh et al. 2016)	53.07	51.02
NGRAM-LS-1	61.29	61.25
NGRAM-LS-2	60.44	58.04
NGRAM-LS-3	**62.40**	**61.98**
BidaF-1	60.03	57.27
BidaF-2	58.44	60.20

Table 3: The first 12 rows show the performance of our method with different parsers and entailment functions. The last 6 rows show the performance of the baseline methods.

the entailment functions at hand; which is 79.28% with the NGRAM-LS-3 entailment function on the “Question Split” and 84.06% with the NGRAM-LS-2 entailment function on the “Text Split”. The next four rows show the performance with the KDG (Krishnamurthy, Dasigi, and Gardner 2017) parser. The errors made by the parser result in an accuracy drop of ˜5% on “Question Split” and a drop of ˜8% on “Text Split”. However, when the customized-KDG parser is used the accuracy on both the split increases. The best accuracy on “Text Spit” is 77.8% which is within 1.5% of the achievable best with the entailments at hand. The accuracy drop on “Text split” also reduces from ˜8% to ˜3.3%. Among the baseline methods which are shown in the last 6 rows, the best score is achieved by the NGRAM-LS-3 entailment function which is 15.4% less than the best performance achieved by our system on “Question Split” and 18.72% less on “Text Split”.

Conclusion

Developing methods that allow machines to reason with background knowledge with premises written in natural language enhances the applicability of logical reasoning methods and significantly reduces the effort required in building a knowledge based question answering system. In this work we have presented a method towards this direction by using function symbols from ASP together with textual entailment functions. Experiments show the success of our method. However there is still scope for further improvements with the best accuracy being 80.7%. The life cycle dataset will be made publicly available to track the progress towards that.

References

Bos, J. 2008. Wide-coverage semantic analysis with boxer. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, 277–286. Association for Computational Linguistics.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Calimeri, F.; Cozza, S.; Ianni, G.; and Leone, N. 2008. Computable functions in asp: Theory and implementation. In *International Conference on Logic Programming*, 407–424. Springer.

Chen, Q.; Zhu, X.; Ling, Z.-H.; Inkpen, D.; and Wei, S. 2018. Neural natural language inference models enhanced with external knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 2406–2417.

Clark, P.; Cowhey, I.; Etzioni, O.; Khot, T.; Sabharwal, A.; Schoenick, C.; and Tafjord, O. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Clark, P.; Dalvi, B.; and Tandon, N. 2018. What happened? leveraging verbnet to predict the effects of actions in procedural text. *arXiv preprint arXiv:1804.05435*.

Clark, P. 2015. Elementary school science and math tests as a driver for ai: Take the aristo challenge! In *AAAI*, 4019–4021.

Dagan, I.; Glickman, O.; and Magnini, B. 2006. The pascal recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*. Springer. 177–190.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *European Semantic Web Conference*, 273–287. Springer.

Flanigan, J.; Thomson, S.; Carbonell, J.; Dyer, C.; and Smith, N. A. 2014. A discriminative graph-based parser for the abstract meaning representation.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6(3):1–238.

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

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

Havur, G.; Ozbilgin, G.; Erdem, E.; and Patoglu, V. 2014. Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 445–452. IEEE.

He, L.; Lee, K.; Lewis, M.; and Zettlemoyer, L. 2017. Deep semantic role labeling: What works and whats next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 473–483.

Jijkoun, V., and De Rijke, M. 2006. Recognizing textual entailment: Is word similarity enough? In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*. Springer. 449–460.

Khot, T.; Sabharwal, A.; and Clark, P. 2018. Scitail: A textual entailment dataset from science question answering. In *Proceedings of AAAI*.

Krishnamurthy, J.; Dasigi, P.; and Gardner, M. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 1516–1526.

Levesque, H. J.; Davis, E.; and Morgenstern, L. 2012. The winograd schema challenge. In *KR*.

Lierler, Y.; Inclezan, D.; and Gelfond, M. 2017. Action languages and question answering. In *IWCS 201712th International Conference on Computational SemanticsShort papers*.

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

Mitra, A., and Baral, C. 2016. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In *AAAI*, 2779–2785.

Parikh, A. P.; Täckström, O.; Das, D.; and Uszkoreit, J. 2016. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*.

Rajpurkar, P.; Zhang, J.; Lopyrev, K.; and Liang, P. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Richardson, M.; Burges, C. J.; and Renshaw, E. 2013. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*, volume 1, 2.

Seo, M.; Kembhavi, A.; Farhadi, A.; and Hajishirzi, H. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.

Sharma, A.; Vo, N. H.; Aditya, S.; and Baral, C. 2015. Towards addressing the winograd schema challenge-building and using a semantic parser and a knowledge hunting module. In *IJCAI*, 1319–1325.

Wang, Y.; Berant, J.; and Liang, P. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, 1332–1342.

Wang, Y.; Lee, J.; and Kim, D. S. 2017. A logic based approach to answering questions about alternatives in diy domains. In *AAAI*, 4753–4759.

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