Written Preliminary Examination-II
Report

Domain-Independent Semantic Understanding of Natural Language

Ben Zhou

Department of Computer and Information Science
University of Pennsylvania

Prepared for the WPE committee and everyone in the NLP community

September 27, 2023

Abstract

Natural language processing (NLP) aims to parse or map natural language utterances to a machine-readable form for computational execution. For a classic computer that works on binaries, a machine-readable form is often a program composed of a limited set of symbols. NLP is challenging in many aspects, but one of the main difficulties is that the space of natural language utterances is exponential to the vocabulary size, making mapping to symbolic spaces non-trivial. This challenge calls for a dedicated research direction in NLP, namely semantic understanding, which is to study how to map utterances to controlled spaces often with the help of human knowledge and statistical models. In this report, we view such controlled spaces to be usually much smaller than the space of natural language. As a result, it provides a more accessible and more accurate execution of the semantics to achieve the specific purposes of the original utterance. However, semantics are often domain-dependent: many systems work best in understanding the utterance when in-domain end-to-end supervision is provided. While such methods attracts much attention through real-world applications, from a research perspective, it is more appealing to study how semantic understanding can be conducted in a generic domain-independent manner that is closer to human-like reasoning. This report looks closely at three papers discussing how mappings between natural language utterances and symbolic processes should or could be considered with domain-independence. We first look at an early philosophical framework that attempts to represent world knowledge in symbolic programs (McCarthy and Hayes, 1969). The framework discusses the necessity and possibility of moving from a free-form domain to a controlled space of operations for more accessible computation. It inspires later work that put symbolic-semantic parsing to work in real-world applications. We discuss one of such early works that learn to find a symbolic program corresponding to the semantic from responses (Goldwasser and Roth, 2013). Lastly, we examine text modular network (Khot et al., 2021), a more recent work that considers pre-trained language models (Devlin et al., 2019) and directly uses natural language to replace the symbols and achieve good results on complicated question-answering tasks. We further extend such thoughts and formulate a theoretical framework for domain-independent semantic understanding, and we hope that it will guide future research in this direction.

Contents

1 Introduction

A general goal of natural language processing (NLP) is to let machines understand human language. This essentially maps a sequence of tokens from a pre-defined vocabulary to an executable form that machines can directly read. There are typically two general types of systems that achieve such a task. First, some popular methods rely on word embeddings, neural networks, and end-to-end learning to approximate specific aspects of the input sequence, such as entities and relations. However, such systems often require prior knowledge of what task they are solving since they do not attempt to interpret the full semantics of the input natural language utterance. As an alternative, some methods incorporate semantic understanding and map the input utterance to an intermediate symbolic program as a generic representation of the input before the program is executed to achieve specific goals.

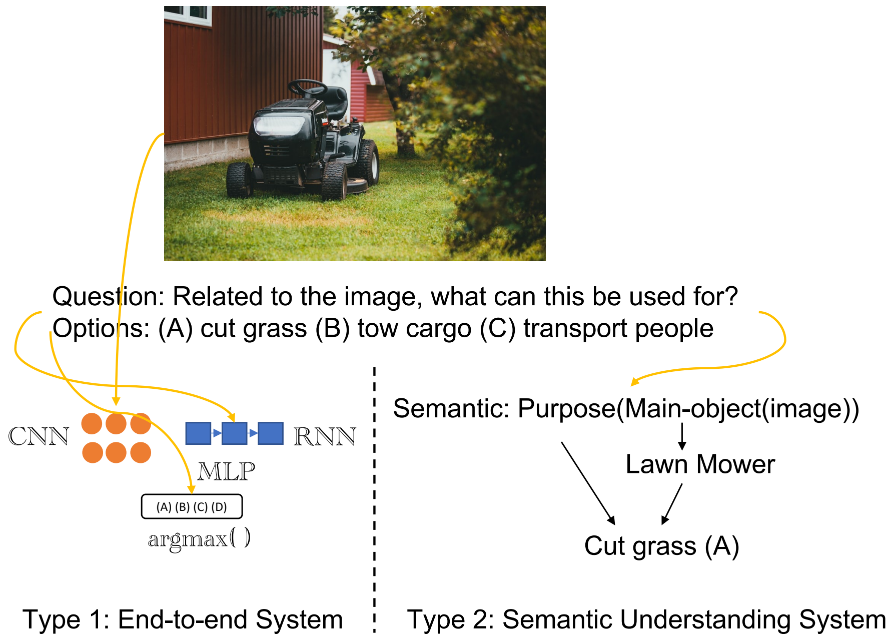
Question: Related to the image, what can this be used for?
Options: (A) cut grass (B) tow cargo (C) transport people
CNN
RNN
MLP
(A) (B) (C) (D)
argmax( )
Semantic: Purpose(Main-object(image))
Lawn Mower
Cut grass (A)
Type 1: End-to-end System
Type 2: Semantic Understanding System

Figure 1: An illustration on the differences between an end-to-end NLP system and a system that relies on intermediate semantic interpretation.

Fig 1 shows a illustrative comparison between the two types of NLP systems mentioned above. Type 1 systems (on the left of Fig. 1) create executable mappings from numerical representations of the input utterances to probability distributions of task-specific labels. When these distribution mappings are estimated by a large amount of direct supervision data, they have been shown to perform well across many tasks, such as named entity recognition and multiple-choice question-answering. However, through numerous benchmarks, researchers have found that such end-to-end systems do not transfer well across domains and tasks. One main reason is annotation artifacts: models, especially modern-day systems that use millions of parameters, are pruned to side-channel signals. In other words, many end-to-end models get the correct answer for the wrong reasons if the data contains such patterns that are often subtle to human readers. One primary reason of such limited transferability is the lack of symbolic structures (e.g., explicitly checking if the word is capitalized in named entity recognition), which humans rely a lot in many problem solving. To address this, one line of the research has been trying to augment the type 1 systems with symbolic structures (Andreas et al., 2015) that mimic such explicit processes. While such work often bring performance gains, it is still very difficult to know what exactly these symbolic structures are learning or memorizing.

This evidence implies that the first type of NLP system may not truly understand the semantic goals of the input utterances while achieving good performances on many tasks. On the contrary, the second type of NLP system (type 2 systems on the right of Fig. 1 attempts to build interpretable and explainable systems by mapping input utterances to a controlled space with human-readable operations and symbols. Comparing to type 1 systems, type 2 systems have two significant benefits. The first is that intermediate representations in controlled spaces introduce easier and more reliable executions for final denotations. For example, identifying the main object in the provided image is a much simpler task than predicting the end-to-end answer, and consequently comes with much smaller room for mistakes. The second benefit is that it decouples multiple aspects involved in a reasoning process (e.g., operation, knowledge, constraints), so it works on different domains simply by switching some components, such as the knowledge base. These two benefits make semantic understanding more appealing for general reasoning than end-to-end supervised learning.

This review will focus on the type 2 systems shown in fig. 1 and look at three works that provide gradual inspirations for semantic understanding. The first one is the McCarthy framework (McCarthy and Hayes, 1969), which we discuss in detail in §2. It provides philosophical perspectives on how knowledge can be or should be represented symbolically. While it does not discuss how to find or map to these symbolic programs given natural language utterances, it sheds light on the robustness of symbolic programs for world knowledge representation and inspires many future works that design real-world applications. Among these future works, one of the early working systems is the learning from responses scheme (Goldwasser and Roth, 2013), which we cover in §4.2. The learning from responses framework learns to parse natural language utterances into pre-define symbolic program space based on binary feedback indicating whether a parsed program was successfully executed or not. It is one of the earliest works that show natural language can be mapped to program space within a controlled setting. However, the limitations are obvious, and one of the most significant ones is that it requires the symbol space to be small enough. This is problematic since the search space grows exponentially with the size of possible symbols, and there are much more symbols to consider in broader domains and applications. We discuss a third work, text modular network (Khot et al., 2021) in §5 that aims to address this problem. It relies on modern pre-trained language models (Devlin et al., 2019) and represents each symbol directly with natural language. In other words, text modular network decomposes complex utterances into a program that is a composition of operations on top of the results from short utterances and simpler questions.

In §6, we connect these three works and discuss existing approaches' limitations in general. We list several open challenges when designing a semantic understanding system in general and present a theoretical framework that meets all critical virtues of individual sub-components. While such frameworks may not be realized in an ideal form soon, we conduct a proof-of-concept experiment on StrategtQA (Geva et al., 2021), a question-answering dataset that is particularly challenging for semantic understanding systems but hugely benefit from successful semantic interpretations. We build a naive system that parses questions into factually-verified sub-questions and show that such an augmentation improves over existing end-to-end systems.

2 The McCarthy Framework

2.1 Overview

This paper is an early positional work that proposes several thoughts on what it means for a program to be able to "reason". In other words, instead of proposing programs that solve individual tasks that are intelligently challenging to human beings (e.g., chess), the author wants to study the general mechanism of what makes a program intelligent, seeking for generalization across problems and tasks. The paper then proposes a self-explanatory framework that defines many aspects of a program that needs to be intelligent.

Although this work focuses on artificial intelligence in general, it is strongly related and inspires the community of natural language understanding (NLU). This work proposes that there exists a set of atomic and symbolic operations which artificial intelligence systems internally use to represent knowledge, situations, and reasoning paths. As noted in §1, the type 2 systems aim to produce an interpretable intermediate representation in a controlled space from natural language utterances. The McCarthy framework introduces a potential way on how this controlled space may look like, that is, with pre-defined sets of operators and symbols and first-order logic chains.

2.2 From a Philosophical Standpoint

The first part of (McCarthy and Hayes, 1969) discusses some standpoints of a general representation of intelligent machines and notions that are adequate to explain "can", "causes" and "knows". The three keywords are seemingly simple and straightforward, but they actually form a very generic view of artificial intelligence, which McCarthy uses to explain how his framework is sufficient to achieve many tasks.

Definitions of Intelligence. (McCarthy and Hayes, 1969) argues that Turing's method of checking whether an intelligent system is truly intelligent through half an hour of a phone conversation with an actual human is impractical for real-world applications, as it will be subject to human limitations. Instead, he proposes to view intelligent systems as "fact manipulators" so that we can check if an intelligent system is sufficient by asking it to solve particular classes of questions that require a sufficient representation of the world. As a result, McCarthy defines intelligence in two parts:

The epistemological part requires systems to have a representation of the world so that solutions can be acquired from the facts expressed in this representation.

The heuristic part requires systems to understand what to do: what facts to use and how to use them when approaching problems and questions.

The two theoretical parts together formulates what McCarthy believes to be sufficient for intelligent machines, which also resembles our definition for semantic understanding. (McCarthy and Hayes, 1969) is devoted entirely to the epistemological part, which we view as a critical component for domain-independent semantic parsing in general. The key capabilities that (McCarthy and Hayes, 1969) want a world representation to have include:

1) Incorporate specific observations and new scientific laws,

2) Identify key entities, such as mathematical symbols,

3) Generalize to any knowledge types,

4) Efficiently store and represent knowledge.

2.3 Reasoning and Representation of the World

McCarthy defines a theoretical program called a reasoning program (RP) that represents general intelligence. A reasoning program perceives the world and interact with people through input/output forms such as languages, visions, and sounds. During such interactions, the RP uses some internal representations for the current scenario, execute certain sub-programs with its world knowledge, and provide intelligent outputs or make corresponding decisions. There are many possible forms of such internal representations, such as RGB-arrays to represent images, temporal sequences to represent scenarios, and discrete classification labels for rules and knowledge. Among these forms, McCarthy argues that there is a clear dominating form, which is ω-order logic with symbols written in natural language. As we have shown in Fig. 1, the Type-2 system relies on such internal representations, where the solution to the given question can be expressed as first-order connections between symbolic functions with a clear natural language phrase as the descriptor.

A reasoning program requires both epistemological and heuristic capability. That is, as shown in Fig. 1, a RP needs to first realize that to solve the question it must find the main object of the image (epistemological) and then to actually find the main object from the image (heuristic). Because the McCarthy framework only focuses on the heuristic part, this paper does not describe how to build a reasoning program, but rather a Missouri program. A Missouri program is a theoretical program that does not try to find proof to achieve a goal but rather checks the correctness of the provided proofs. It is also able to execute the given proof or a given strategy. With the type-2 system example shown in Fig. 1, a Missouri program does not attempt to find the right program, but rather checks if the given program is valid with respect to the question.

Discussing a Missouri program allows us to only focus on the design of how the world can be adequately represented, but not how to perform reasoning. (McCarthy and Hayes, 1969) argues that there are several ways to do this. A metaphysically adequate representation uses high-level but implicit notions. Such representations are generic descriptions on how the world operates. For example, we can represent the world as movements and interactions among countless of particles or atoms. We can also use an extensive set of automata to represent the world, where each sub automata receives states from other automata or at time t, and decides its state at time $t+1$. Metaphysically adequate representations are seemingly appealing since they can explain any possible scenarios that may occur in the real world. However, there are many practicality issues that come with them. For example, Automatas would require a very large set of states to represent situations and a dynamic interconnection between graphs. It would be almost impossible to put such automatas into applications that involve multiple agents, since it is very hard to represent a person with automata, which would require the automata to list all the internal states and find ways to extract values.

Due to the many issues with metaphysically adequate representations, McCarthy believes that an epistemologically adequate representation is more appealing in practical applications. Epistemologically adequate representations are representations that are enough to describe certain aspects or knowledge that a particular agent needs or has, and the are not required to be sufficient for any possible scenario. Natural language is one example of epistemologically adequate representations, since it is enough for people to communicate, but insufficient to represent much more complicated and implicit tasks such as how we recognize faces. Later in this report, we borrow this idea and discuss how natural language can serve as an excellent representation for generic semantic understanding.

2.4 Formalism

Due to the many impractical reasons for metaphysically adequate representations, it is wise to research epistemologically adequate representations instead. (McCarthy and Hayes, 1969) describes one kind of epistemologically adequate representation using first-order logic and short natural language phrases as atomic operators. We describe the notions of this representation

below.
Situations. The main component in this framework is called situations, which is the complete state of the universe at an instant in time. In practice, a system's perception of the situation will not be perfect, but only composed of facts about the situation. These facts are often interrelated, so more facts can be derived about current or future situations. In other words, we need the situation representation to be capable of understanding "what-if" relations based on current information. In order to provide partial information to situations, (McCarthy and Hayes, 1969) introduces a notion called "fluent".
Fluents. As situations can not be described completely, and the author introduces the notion of fluents to make partial descriptions possible. Specifically, a fluent function evaluates some properties of the situation and makes an assertion as the output. For example, given a situation s, a person p in place x, and x is raining, we can use two fluents and their conjunction to describe this scenario, namely $at(p,x)(s) \wedge raining(x)(s)$. There are several other possible ways to write this fluent, such as:

$$at(p,x,s) \wedge raining(x,s)$$

$$[at(p,x) \wedge raining(x)](s)$$

$$[\lambda s\prime.at(p,x,s\prime) \wedge raining(x,s\prime)](s)$$

There are some essential examples of fluents. $time(s)$ determines the current time associated with the situation, $in(x,y,s)$ expresses that x is in location y in situation s. Transitive law can be applied to this fluent, and we can write it as

$$\forall x.\forall y.\forall z.\forall s.in(x,y,s) \wedge in(y,z,s) \rightarrow in(x,z,s)$$

These fluents are typically associated with natural language phrases such as "John is at home", but different phrases with the same semantic under specific situations will be expressed similarly. Such a reduction will help the reasoning system's understanding.
Causality. As the name suggests, causality is introduced to make connections between fluents, so we can represent rules such as the situation s will be followed by the current situation $s\prime$, given both situations are described as fluents, and the current situation satisfies the conditions defined by the fluents.

For example, we can define an operator $F(\pi,s)$, so that for some situation $s\prime$ in the future of s, $\pi(s\prime)$ holds. This can also be defined with a situational fluent $next(\pi)$ that represents the next situation $s\prime$ in the future of s for which $\pi(s\prime)$ holds. Under such notation, we can translate the natural sentence "By the time John gets home, Henry will be home too" as

$$at(Henry, home(Henry), next(at(John, home(John)), s))$$

However, the definition of causality varies from different philosophical views. This form of representation above does not enforce a real causation relationship, as Henry getting home might not be causally related to John getting home. A more rigorous operator should define that $\pi(s\prime)$ hold for all situations $s\prime$ following s, which can be written as $G(\pi, s)$, as defined in (Prior, 1968).

Actions. Actions are another class of constituents, that can be defined with the help of a special fluent $result(p, \sigma, s)$. $result(\cdot)$ outputs a situation that is the result if p carries out action σ, starting in situation s. With the help of actions, a system is able to express certain laws of abilities. For example, define an action called $opens(item, key)$, we can define the the ability of opening something with its key requires having the right key, carried out by the action of $opens(\cdot)$ as:

$$has(k) \wedge fits(i, k) \rightarrow open(i, result(opens(i, k)))$$

Assuming that the item is a safe sf, and under a specific situation s that requires a person p to be physically at the safe's location in order to open the safe, this ability can also be written as:

$$has(p, k, s) \wedge fits(k, sf) \wedge at(p, sf, s) \rightarrow open(sf, result(p, opens(sf, k), s))$$

Strategies. Strategies are a sequence/combination of actions, so we can express an event that comprises multiple actions. For example, we can represent a natural language description of "move the box under the bananas, climb onto the box and reach for the banans" as

$$\textbf{begin } move(Box, under - box); climb(Box); reach(Bananas) \textbf{ end}$$

In a broader scope, we can also use "if-then" clauses to represent general algorithms.

Knowledge. The author views knowledge as a series of causality relations that involve actions and strategies. For example, knowing that under some situation s, a person needs the right key and to be at the right location in order to open a physical safe as described above is a type of knowledge.

In other words, knowledge is viewed as a set of rules indicating how situations can be described, and how situations relate to each other with respect to different strategies that may happen. There are several limitations that come with such definition for knowledge. One significant limitation is that the McCarthy framework does not allow numerical probabilities to be considered. While most uncertainties and probabilities arise because we cannot perfectly describe the situation with enough fluents, there are probabilities that are natural uncertainties (e.g., lotteries) and should be considered in knowledge representation.

3 Impacts for Future Work

In this section, we describe the McCarthy framework, a theoretical framework that provide insights into the general semantic understanding process that we focus in this report. Some key perspectives are:

1. Intelligent systems show intelligence by producing proper outputs given inputs. This process requires the system to have proper internal representations.

2. A metaphysically adequate representation such as particle movements or automata is generic, but impractical. On the contrary, epistemologically adequate representations are those that are only sufficient for particular purposes, but are more interpretable.

3. Short constituents with natural language namings composed with first-order logic can be used to construct a epistemologically adequate representation that formulates how states, actions and knowledge relate to each other.

In the next sections, we introduce two works that build upon similar thoughts. In §6, we discuss how we borrow the McCarthy philosophies to propose generic reasoning framework for natural language queries.

4 Learning from Natural Instructions

The McCarthy Framework is among the first theoretical works to try to connect natural language to a controlled space of symbolic functions and first-order logic. As described in §2, the McCarthy framework primarily focuses on the epistemological part of intelligence, which discusses how systems should represent the world, facts, and rules to facilitate real-world problem-solving. However, the McCarthy framework does not discuss how such representations can be acquired and applied to actual problems.

In this section, we discuss the framework of learning from natural instructions introduced by Goldwasser and Roth (Goldwasser and Roth, 2013), which not only uses a similar internal representation presented in §2, but also extends the focus to the heuristic part of learning that automatically learns which rules to use.

4.1 Overview

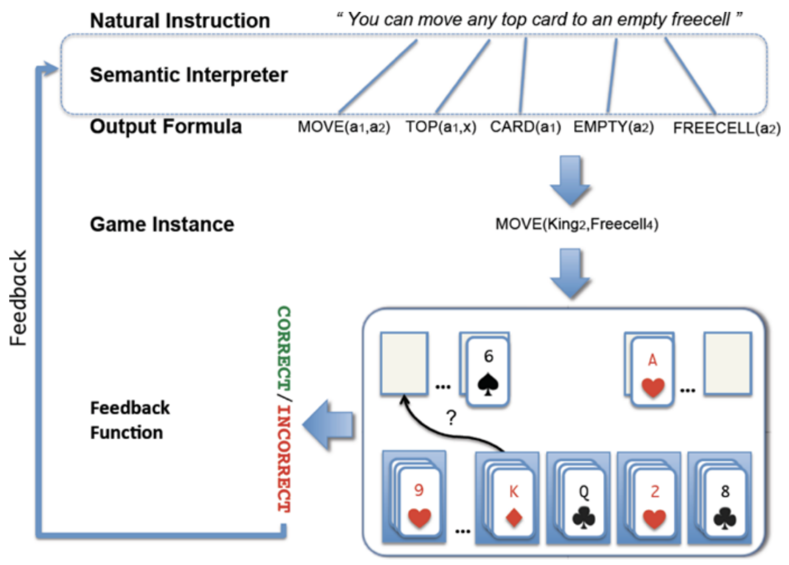

Figure 2: An example of learning from natural instructions.

The McCarthy framework argues that we need a representation of world knowledge but does not specify how an artificial intelligence agent may acquire that world knowledge, nor select what knowledge to use given different tasks. It is appealing to ask human annotators to write everything down in a machine-readable symbolic format. For example, if we want to represent rules for a card game, such as when a card can be moved to where, we can write it with a few simple symbolic functions, similar to the representation proposed in §2:

$$Move(a_1, a_2) \leftarrow card(a_1) \wedge top(a_1, x_1) \wedge freecell(a_2) \wedge empty(a_2)$$

With careful definitions and implementations of each function, an automatic agent may successfully interpret this rule. However, asking humans to provide card game rules in such a format is much more complicated than the natural language instruction corresponding to this rule, which is as straightforward as

You can move any top card to a free cell if it is empty.

According to Goldwasser and Roth, this implies that we should try to build a system to learn to understand instructions in natural language instead of asking human annotators to write every possible rule. This is because using human annotations of machine-readable representations to solve every possible task is too costly to be feasible. In contrast, natural language instructions are much easier to acquire, since humans use natural languages to define different tasks all the time, in order to teach someone who is not familiar with the specific task. This suggests that natural language instructions for most common tasks should already exist; if not, it would not be hard for one to come up with instructions in natural language.

This work, with an over-viewing example shown in Fig. 2, follows this intuition and proposes one of the earliest approaches to not only represent, but also learn world knowledge in symbolic formats from natural language in real applications. The proposed semantic understanding algorithm learns to parse instructions by letting the semantic parser to trial and error on a few real-world responses to iteratively correct current hypotheses and propose better ones. With the card game example, this means that the algorithm checks whether the current learned symbolic parsing for the card game rules is correct by making predictions on whether some card moves are valid or not, and it compares its predictions to the actual labels to make corrections until all validity predictions are correct.

4.2 Learning Instructions from Binary Feedbacks

(Goldwasser and Roth, 2013) formulates the problem of mapping natural language inputs to a symbolic program as an optimization problem to find

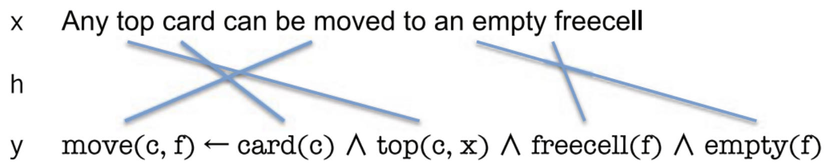
x
Any top card can be moved to an empty freecell
h
y
move(c, f) ← card(c) ∧ top(c, x) ∧ freecell(f) ∧ empty(f)

Figure 3: The input sentence x, hidden mapping h, and output symbolic program y on the card game example.

an alignment h between natural language fragments and symbolic operators, and output program y, given input sentence x. The optimization involves a feature function $\phi(\cdot)$ and a learnable weight W. After W is learned, the prediction function can be hence written as

$$y = \text{argmax}_{h,y} W^T \phi(x, h, y)$$

With the card game example, we can sketch the mapping h in Fig. 3, where some natural language fragments are mapped to symbolic programs based on lexical similarities. Even though lexical challenges are greatly minimized in this example, there is also the challenge of composition, which determines how each symbol should be composed together into a program.

Because (Goldwasser and Roth, 2013) assumes that there is no annotated symbolic program available for training, they propose to use binary real-world responses (e.g., in the card game example, we can check whether a card move is legal using the learned rules) to check whether a current hypothesis is correct, and further update the learnable parameters to make the final program consistent with all real-world instances. More specifically, (Goldwasser and Roth, 2013) assumes that the learning algorithm has access to a set of binary feedback; a positive feedback means that the current learned program successfully handled a real-world instance such as a card move, while a negative feedback means that the current learned program is not entirely correct, as the correct parsing would successfully predict the validity of all possible instances.

Learning only from such binary feedback is particularly challenging, as the parsing algorithm is a structural prediction problem. In order to update part of the structure, one needs to find a "fault-assignment" method, which determines which part of the structure is responsible for any mistakes and further updates the structure to avoid future similar mistakes. In the next section, we describe the specific algorithm that Goldwasser and Roth propose.

Learning Algorithm. For any hypothesis parsing that yields positive feedback, the algorithm adds that to a set of “gold” training instances because such programs are consistent with the world observation. Things get more complicated when a hypothesis program produces negative feedback. We can choose not to do anything or update the current weight vector uniformly so that the weight is discouraged from producing the same program given the input sentence in the future. However, none of these two are good enough. Not updating anything would waste all negative examples, while updating weights uniformly is not intuitively sound as only part of the structure is relevant to the input sentence. To solve this, the authors propose to use a combination of algorithms to approach this learning task. If the algorithm has seen a “correct” program for the input x that the parser makes a mistake on (i.e., such program exists in the “gold” training instance set), it will make a structural update to the weight vector by

$$w = w + \phi(x, h^*, y^*) - \phi(x, h, y)$$

where $h*, y*$ are from the “gold” program, and h, y are from the current program that makes a mistake. When there is no such “gold” program from previous observations, the algorithm updates the weight vector uniformly, that is

$$w = w - \phi(x, h, y)$$

“What is the population of the largest city in New York?”
***Output*:**
`Population( Largest (City (`**`NEW_YORK_CITY`**` ))).`

Figure 4: An example of an incorrectly predicted program with partially correct structures.

The uniform update when no existing structure is present is still less ideal because there are simple heuristics that we can use to let the model only update part of the parameters. Fig. 4 shows an example of a partially correct predicted program. Most symbolic functions such as population, largest, and city are correct, with the only mistake in interpreting the state of New York as New York City. Intuitively, we would want to update only the New York City part and avoid penalizing all other structures that are correct. To do this, the authors assume that the algorithm has control over the conditional probability over individual text-to-symbol mapping, such as

$$p(\text{NEW\_YORK\_CITY} | \text{“}New\ York''})$$

Then, the uniform weight update can be changed to

$$w = w - \phi(x, h, y) \cdot (1 - p(y_i|x_i))$$

This means that the penalty will be stronger on less certain alignments between text fragments and symbols, based on the intuition that it is less likely for high-confidence mappings to be wrong. The authors compute an estimation of the text-to-symbol probability by running all existing structures that have positive feedback and aggregate their decisions.

4.3 Mapping Text Fragments to Symbols

In this section, we describe how (Goldwasser and Roth, 2013) uses the learning objectives in §4.2 to build an inference system that learns to map textual fragments to symbols. The overall framework uses integer linear programming (ILP). This allows the inference process to consider the entire program space by injecting human knowledge and learned features.
Objective Functions.

$$F_{\mathbf{w}}(\mathbf{x}) = \arg\max_{\alpha,\beta} \sum_{c \in \mathbf{x}} \sum_{s \in D} \alpha_{cs} \cdot \mathbf{w_1}^T \Phi_1(\mathbf{x}, c, s) + \sum_{c,d \in \mathbf{x}} \sum_{s,t \in D} \sum_{i,j} \beta_{cs^i,dt^j} \cdot \mathbf{w_2}^T \Phi_2\big(\mathbf{x}, c, s^i, d, t^j\big)$$

Figure 5: The main objective function used by the ILP inference.

This ILP problem aims to optimize a linear equation under several linear constraints. The main linear equation to be optimized is described in Fig. 5. Specifically, it maximizes two different sets of weights and feature vectors. The ones associated with α determines whether a textual constituent c is mapped to symbol s (denoted as α_{cs}. The ones associates with β determines whether the the i^{th} argument in symbol s (from constituent c) and the j^{th} argument in symbol t (from constituent d) are the same (denoted as β_{cs^i,dt^j}). In short, this objective function is trying to find a set of α and β assignments according to the weights and feature functions under a set of constraints.
Constraints. We need the constraints because the system should not simply assign all α and β to be 1 to achieve a clear maximization, as there are clear limitations on when we can assign certain values. There are several other constraints, but we will only list some examples here.

For example, a constituent can only be assigned to one symbol, written as

$$\forall c \in x, \sum_{\alpha_{cs}} \leq 1$$

If two symbols share a variable, these two symbols must be active:

$$\forall c, d \in x, s, t \in D, \beta_{cs_i,dt_j} \implies \alpha_{cs} \wedge \alpha_{dt}$$

4.4 Features

In this section, we describe how the feature vectors $\phi_1(\cdot)$ and $\phi_2(\cdot)$ is designed in (Goldwasser and Roth, 2013).

$\phi_1(\cdot)$ are features that control the lexical mapping from textual fragments to symbols, called alignment decision features. Such features are created with manual lexicon alignment (e.g., "card" to $card()$) and WordNet (Miller, 1992) similarities. The mapping lexicon is also dynamic: whenever a mapping not in the lexicon appears to produce positive feedback, that mapping will be added to the lexicon.

$\phi_2(\cdot)$ are called compositional decision features, as it determines how β should be assigned. These features are based on the dependency tree (Klein and Manning, 2002) of the input sentence.

4.5 Experiments

The primary experiments are conducted on a few card game instructions, with each card game provided with ten labeled game instances: 5 positive and five negatives for feedback purposes. On the card game example used in previous sections, the initial model (before any iterations of learning over the feedback) achieves an accuracy of 76%. In contrast, the learned model gets 95%, which is almost the correct rule for different kinds of descriptions.

5 Text Modular Networks

The McCarthy framework (McCarthy and Hayes, 1969) provided high-level intuitions on how symbolic representations can be used to represent world knowledge. The learning from natural instructions model (Goldwasser and Roth, 2013) put much thought into an application by building an ILP-based inference with learnable weights from real-world feedback. While both work and their related papers provide solid intuitions and inspirations to the field of natural language processing, their limitations are apparent, especially from the perspectives of recent breakthroughs with large pre-trained language models based on transformers (Devlin et al., 2019).

One of the main limitations is that the symbolic representations in both works only minimally consider how to find the symbolic program to solve the input query. Moreover, they only work on queries where the objective is explicit, such as direct instructions or direct queries like "find the largest city". If the query is fuzzy or requires contextual understanding, such parsers would generally fail. At the same time, recent tasks that consider complex problem-solving have made manually designing symbols and their compositions less feasible. For example, we may need millions of new operators to cover new concepts such as "the number of field goals in a new sport".

As a result, some recent approaches aim to tackle tasks that require contextual understanding in an end-to-end fashion with deep neural networks. While some of these methods are effective, they are flawed when the task requires some level of symbolic reasoning. This is because most large models suffer from annotation artifacts and only learn inductive biases: they do not solve the tasks for the right reason. This is shown by many work that tries to interpret large models' decisions. Unlike a symbolic program in (Goldwasser and Roth, 2013), many large models provide an unreasonable solving process while still getting the final answers correct.

To this end, people have been trying to build deep neural models with symbolic structures (Andreas et al., 2015), but such work is still not fully-interpretable and often exploits biases and artifacts. In this section, we look at a paper with a more symbolic process but represent symbols directly with language. Consider if the system is to answer the question

What country is the person that made the most field goals from?

We can write this in a symbolic format with specific operations such as

$$\text{origin}(\text{argmax}(\text{num-field-goals}(\cdot)))$$

As one could imagine, parsing and executing this function to derive the final answer is quite difficult because a system needs to include all operations such as num-field-goals($\cdot$) in its operation space, successfully parse the query, and execute all operations to the correct values. However, it will be much easier if we conduct symbolic reasoning only to parse the original question into simpler ones and then answer each simple (often called one-hop) question with neural models and their inductive biases. This is based on the intuition that many symbols can be written as a natural language straightforwardly. For example, the execution of symbols such as num-field-goals($\cdot$) can be approximated by answering a question of "How many field goals are there?". Such simple and straightforward questions can almost be perfectly solved by the inductive biases from existing deep models while keeping an interpretable process of reasoning in the form of asking multiple questions and composing their answers.

5.1 Overview

In this section, we introduce a framework called text modular network (Khot et al., 2021). This framework learns to ask decomposed questions to solve complex reasoning tasks. Fig. 6 shows two example questions that can be better approached with text modular network. Instead of symbolically design an execution program, this framework applies symbolic thinking to the question decomposition step. This paper first categorizes questions into several symbolic forms based on the questions' reasoning types. From these symbolic forms, the authors propose an automatic system that finds related questions whose answers can be composed to the original question's answer according to the reasoning type. With the questions and their corresponding sub-questions, this paper trains a generative model called *next question generator*, which relies on the framework that produces sub-questions for each test question and composes their answers.

5.2 Next Question Generator: Supervision Collection

Text modular network relies on a crucial component that asks "next" questions (which are usually simple and straightforward). In Fig. 6, for example, this next question generator decomposes the complex question of "how many years did it take for the services sector to rebound" into a sequence of easier questions: "in what year did the service sector rebound", "when did the services sector start to take a dip" and a difference calculation. Such decomposition is intuitive, but training a model that can produce it is difficult

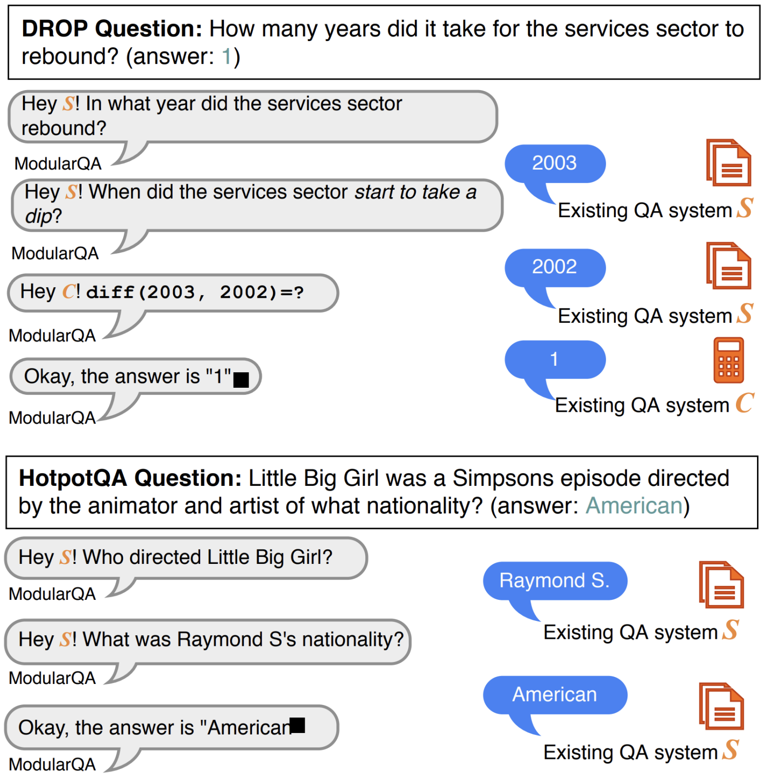
DROP Question: How many years did it take for the services sector to rebound? (answer: 1)
Hey S! In what year did the services sector rebound?
ModularQA
2003
Existing QA system S
Hey S! When did the services sector start to take a dip?
ModularQA
2002
Existing QA system S
Hey C! diff(2003, 2002)=?
ModularQA
1
Existing QA system C
Okay, the answer is "1"
ModularQA
HotpotQA Question: Little Big Girl was a Simpsons episode directed by the animator and artist of what nationality? (answer: American)
Hey S! Who directed Little Big Girl?
ModularQA
Raymond S.
Existing QA system S
Hey S! What was Raymond S's nationality?
ModularQA
American
Existing QA system S
Okay, the answer is "American"
ModularQA

Figure 6: An example question-answering pipeline using text modular network. The framework decomposes complex or multi-hop questions into simple or one-hop sub-questions and uses dedicated modules to answer sub-questions and recompose them into a final answer.

because there is no sufficient training data available.

This is where symbolic reasoning comes in for this work. The authors propose a way to find sub-questions using the gold answer as a signal automatically. This process is composed of several steps, which are introduced below.

Assumptions. The authors make a few assumptions about the questions they plan to solve within the paper. They employ two question-answering models to answer simple questions: one trained on SQuAD (Rajpurkar et al., 2016), and another one that symbolically handles mathematical calculations, similar to a calculator. A modern approach based on pre-trained language models can do very well on extractive question-answering after it is trained on large datasets such as SQuAD. This is one of the key assumptions supporting the overall framework. Based on what a SQuAD QA model and a calculator are adequate to solve, this paper focuses on two datasets: DROP (Dua et al., 2019) and HotpotQA (Yang et al., 2018).

Decompose Questions as Supervision. Based on the QA modules and the two datasets, the authors propose an automatic question decomposition method that only works when a gold answer to the question is given. As a result, this method cannot be used directly for test-time inference but can be applied to acquire training decomposition instances. This method is based on educated guesses on how a question may be solved. This educated guessing mechanism first categorizes questions into five reasoning types based on keywords:

- **Difference.** *How many days before X did Y happen?* Such questions compare the difference between two numbers. There usually exist two numbers in the given context that produces a difference equal to the gold answer.
- **Comparison.** *Which event happened before: X or Y?* Such questions compare certain numerical properties of two entities. We should look for numbers or dates mentioned near the entities that can form a comparison.
- **Complementation.** *What percent is not X?*
- **Composition.** *Where was 44th President born?*
- **Conjunction.** *Who acted as X and directed Y?*

The five reasoning types cover 89% of the training questions. The next step is to find sub-questions for each one of them. This process is conducted in two steps, which we detail below using *how many days before X did Y happen?* as an example:

1. The authors first find all possible hints from the given context based on the assumptions made for each reasoning type. In this case, the system will look for all numbers mentioned in the evidence paragraphs until it finds two numbers that produce a difference equal to the given answer.
2. The authors train a question generator $\mathbb{G}$ from SQuAD contexts to generate SQuAD questions. They then apply $\mathbb{G}$ on related evidence paragraphs until they find two questions, each producing a corresponding number in the previous step.

3. Based on the assumption that if two questions can be asked based on the evidence paragraph and can be answered with two numbers that produce a difference equal to the gold answer, the authors use these two questions as a decomposition for the input question.

This process does not use additional direct supervision, as it relies on distant supervision from a human injection of symbolic knowledge. Compared to previous work (Goldwasser and Roth, 2013) that executes a symbolic program, text modular network approximates many of the straightforward symbolic operations as natural questions but leave the tricky part (i.e., how individual program look like or how operations should be connected) as a symbolic process.

5.3 Experiments

Training. With the distant supervision on question decomposition collected in §5.2, the authors train a BART model (Lewis et al., 2020) to learn to generated sub-questions given a complex multi-hop question, as Fig. 6 shows. Each sub-question is answered by either the SQuAD-based QA model or the calculator module.
Results. Experiments show that compared to DecompRC (Min et al., 2019), text modular network has a 1.3% lower accuracy on HotpotQA. However, human evaluations show that they prefer the decompositions from text modular networks 68% of the time over DecompRC as explanations. This shows that even though the system has a slightly worse overall performance, the decompositions are more accurate on correct instances.

6 Future Work

6.1 Limitations on Existing Work

In this review, we discussed three work that progresses from early philosophical views (McCarthy and Hayes, 1969) to symbolic implementations for real-world applications (Goldwasser and Roth, 2013) and finally to modern approaches that represent diverse symbols directly with natural language (Khot et al., 2021). All three works have a common theme, corresponding to this review's title: semantic understanding. Semantic understanding, in my view, is the process of mapping natural languages to a more controlled space, which can be executed to achieve the goal of the input language more easily. Semantic understanding is very challenging as it involves several design problems.

- **Designing the controlled space.** Controlled space is the intermediate representation systems use to understand natural language queries better. It can be a symbolic program as in (Goldwasser and Roth, 2013), or natural languages directly with some special tokens as in (Khot et al., 2021). This space needs to meet a few key requirements: 1) it needs to be comprehensive enough to represent enough perspectives, operations, and knowledge types sufficiently; 2) it needs to be symbolic enough to provide feasible execution and composition and avoid biases and artifacts; 3) it also needs to be adaptive for new knowledge and new reasoning chains. Early work that relies on symbolic programs often fails requirements 1) and 3) using a pre-defined set of symbols. Recent work often suffers from 2) if they employ natural language directly as the intermediate representation.

- **Mapping natural language to controlled space.** After defining a controlled space, a system needs to find a way to map natural language inputs to the controlled space by correctly parsing the semantics. This is also a particularly challenging step and should ideally have the following properties. 1) The mapping process should be generalizable to all domains, as long as the input query is text-based. 2) It should perform equally well on rare queries (i.e., texts that do not often occur in common natural language. 3) It should be able to produce explanations (which also includes an explanation of why a query cannot be processed) as opposed to a black-box mapping. This is often the weaker part in semantic parsing systems compared to the robustness of common controlled space designs. Early work tends to do better on

3) while modern approaches often fail 3) the most.

- **Executing the controlled representation.** With the natural language utterance mapped to a representation in the designed controlled space, a system would also need to execute the representation to achieve the goal designated by the original utterance. This step clearly depends on the design of the controlled space, so the properties of an ideal execution system largely share the ones of the controlled space design. However, one additional challenge unique to this step is the information retrieval (IR) module. It is often infeasible for the controlled space to include all necessary real-world knowledge, facts, and rules due to the nature of the space being confined. As a result, a system often needs to perform IR to find corresponding knowledge and values for a controlled representation to be successfully executed.

In this review, we will share our thoughts on some directions that may produce better systems that do better on controlled space design and natural utterance mapping, each concerning its own criteria.

6.2 Intuitions and Preliminaries

The process of semantic understanding has a recursive nature. There is usually another level of semantic space from one controlled space that is even more controlled than representations can be mapped to by executing certain symbols. In practice, the benchmarks do not often require this process to be recursive, as the tasks can be sufficiently resolved with a single-level controlled space. However, we view the ideal framework to be a recursion until an atomic operation of a very naive and simple task is reached. To simplify this idea, we formulate this idea as "asking the next question", similar to the one proposed by (Khot et al., 2021), but more general. In the following few sections, we first provide a few motivating examples and then discuss what some of the underlying challenges are and how they may be approached.

6.3 Motivating Examples

Example 1. The system is given an image and then asked "what breed is this?"

To answer this question, without seeing the actual picture, it is natural for humans to ask "what is the main object in the image?" This is a semantic understanding independent of what the image is showing. The semantic understanding process may ask more fine-grained questions depending on the answers. For example, if the image shows a dog, the system should ideally find the distinctive features of different breeds of dogs, such as their fur length, color, and nose shape. As opposed to most systems that encode the image and the question simultaneously, we show that a proper semantic understanding model should decompose the question based on pure text and treat the image as a knowledge base. This ensures that the process is fully interpretable and avoids models using irrelevant information (e.g., the number of legs of the dog, since all breeds of dogs have four legs) as artifacts.

Example 2. The system is asked, "Is the smallest prime number an acceptable number of children in 1980s China?"

This question is not very hard for humans if that person knows China imposed a one-child policy in the 1980s, and the smallest prime number is 2. However, this is a particularly challenging question for NLP systems since systems would need to know about the one-child policy to answer this question. A traditional information retrieval step that relies on keywords has little chance of retrieving the one-child policy concept, as there is much noise such as "prime number". However, we can perform a similar decomposition process based on the pure text by asking the next questions: *what is the smallest prime number?*, *Is there any constraints on the number of children in 1980s China?*. The latter will more precisely locate the one-child policy during the execution step because the semantic understanding process correctly determines the prime number argument to be independent of the one-child policy concept.

6.4 Formulation

We view the semantic understanding process as sequentially generating sub-questions based purely on the text that is independent of the knowledge bases so that they can come from Wikipedia, Google search, images, or any other sources. Formally, given a textual query Q and common sense knowledge $\mathbb{C}$, the semantic understanding model gen_next_question$(\cdot)$ generates a set of sub-questions $\mathbb{S} = \{S_1, S_2, ..., S_n\}$. After generating each S_i, an fact verification model gen_knowledge$(\cdot)$ takes each S_i to a set of knowledge sources $\mathbb{K} = \{Google, Wiki, Image...\}$ and produce an answer A_i. In other words, we have

$$S_i = \text{gen\_next\_question}(S_1, A_1, ...S_{i-1}, A_{i-1}, \mathbb{C})$$

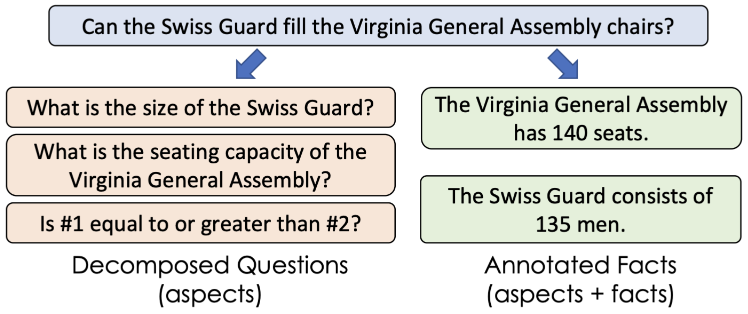
Can the Swiss Guard fill the Virginia General Assembly chairs?
What is the size of the Swiss Guard?
What is the seating capacity of the Virginia General Assembly?
Is #1 equal to or greater than #2?
Decomposed Questions (aspects)
The Virginia General Assembly has 140 seats.
The Swiss Guard consists of 135 men.
Annotated Facts (aspects + facts)

Figure 7: An example question from StrategyQA, along with its annotated decomposed questions and supporting facts.

where

$$A_i = \text{gen\_knowledge}(S_i)$$

We also have a special class of invalid answers, where $A_i = \emptyset$. This implies that S_i is not straightforward enough for gen_knowledge, and the model should learn to produce more fine-grained questions or questions for clarification for the given Q. We assume that the last answer A_{last} is the answer to the original question Q.

It is clear that this formulation is generic to all domains and tasks, whether it's science, history, text-QA, or visual QA. However, there is a big assumption of an excellent commonsense knowledge set $\mathbb{C}$. This set guides the entire parsing process, from the simple ones such as the question in Example 2 is a comparison, to "there may be some laws or constraints limiting the number of children for this question to be valid". Furthermore, after getting specific answers about the one-child policy, $\mathbb{C}$ should interpret "number of child" as "number of child per family", as this is the most likely interpretation. However, an ideal system should ask a follow-up question checking whether it means "per family", as a rigorous gen_knowledge$(\cdot)$ may produce $\emptyset$. The benefit of having a sub-question-based system is that we can observe and understand how and why an answer is derived. The system should make proper assumptions, such as 'number of child" means "number of child per family", but at the same time fully understand that other alternatives exist and some of them will change the final answer.

Additional Information	Accuracy
None	53.3
Decomposed Questions	59.7
Annotated Facts	84.5

Table 1: T5-3B accuracy on StrategyQA dev set when different information is provided for both training and evaluation.

6.5 Experiments

We conduct experiments that make attempts to put the formulation in §6.4 to real-world applications. We use StrategyQA dataset (Geva et al., 2021) as the primary benchmark, as it contains tricky questions such as the second motivating example in §6.3. The strategyQA dataset annotates both "decomposed questions" and "gold facts" for each question as shown in Fig. 7. We use the gold facts to approximate sub-questions because we intend to conduct proof-of-concept experiments. The annotated facts contain more details than the decomposed questions (e.g., the answer type), so they are better for an entailment model to decide the final binary answer. As experiments show in Table 1, having the annotated facts as supporting evidence for an entailment model is much better than using the decomposed questions alone. Using the annotated facts is just another form of the sub-question formulation proposed in §6.4, and it does not alter the assumptions in any way.

We train a sequence-to-sequence model T5 (Raffel et al., 2020) to generate the sub-questions.[1] The decompositions often contain factual errors such as "South Africa is a member of NATO". Most of these facts are trivial enough for a larger language model such as GPT-3 (Brown et al., 2020) to memorize, so we use GPT-3 for factual-error correction. Specifically, we give GPT-3 a prompt of *fix the input sentence with correct facts if there are factual errors* along with a few examples, and GPT-3 can produce a correct statement given each decomposition generated by our T5 model.

Table 2 shows that our system's performance is superior to using T5 and GPT-3 alone, serving as a proof of concept that the framework proposed in §6.4 can be superior to existing end-to-end systems on complicated question-answering tasks.

[1] In this case, the sub-questions are the supporting facts as shown in Fig. 7.

System	Dev	Test
T5-Large	55.9	-
RoBERTa*-IR	65.8	64.9
GPT-3	62.5	64.1
GPT-3 CoT	65.9	63.7
Ours	**70.3**	**67.4**

Table 2: Accuracy on StrategyQA.

6.6 Challenges

Although experiments show early signs that a semantic understanding can be critical in approaching complicated tasks, there are still many open challenges. One of the most important open problems is estimating the commonsense knowledge set $\mathbb{C}$. In §6.5, we do not discuss this issue and conveniently used whatever a T5 model memorizes as $\mathbb{C}$. However, this is evidently not enough, as example 1 in §6.3 cannot be solved by such a commonsense knowledge set. To this end, more research towards symbolically mining $\mathbb{C}$ might be necessary.

7 Conclusion

In this report, we discuss the problem of semantic understanding gradually. In §2, we examine early philosophical views on representing knowledge as symbolic programs. This shows the importance of mapping natural language utterances to a controlled space for easier and more accurate execution. The framework inspires many works on semantic parsing and understanding in real applications, such as learning from responses (Goldwasser and Roth, 2013), which we introduce in §4.2. After large language models (Devlin et al., 2019) are shown to bring deeper and more diverse contextual representations, many works aim to represent symbols in a program directly with natural language. We introduce text modular network (Khot et al., 2021) in §5, which attempts to perform sub-question decomposition for complicated questions. Text modular network features modern language models but only resembles early thoughts on symbolic semantic understanding. We propose a theoretical framework in §6 that extends the thoughts in text modular network to a more generic semantic understanding. We conduct a proof-of-concept experiment that implements our proposed framework and show that it is superior to end-to-end methods on StrategyQA benchmark (Geva et al., 2021). We discuss future directions and inspire future work on symbolically-inspired semantic understanding.

References

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2015). Deep compositional question answering with neural module networks. *ArXiv*, abs/1511.02799.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T. J., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *ArXiv*, abs/2005.14165.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., and Gardner, M. (2019). Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *NAACL*.

Geva, M., Khashabi, D., Segal, E., Khot, T., Roth, D., and Berant, J. (2021). Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *TACL*, 9:346–361.

Goldwasser, D. and Roth, D. (2013). Learning from natural instructions. *Machine Learning*, 94:205–232.

Khot, T., Khashabi, D., Richardson, K., Clark, P., and Sabharwal, A. (2021). Text modular networks: Learning to decompose tasks in the language of existing models. *NAACL*, page 1264–1279.

Klein, D. and Manning, C. D. (2002). Fast exact inference with a factored model for natural language parsing. In *NIPS*.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine intelligence*.

Miller, G. A. (1992). Wordnet: A lexical database for english. *Commun. ACM*, 38:39–41.

Min, S., Zhong, V., Zettlemoyer, L., and Hajishirzi, H. (2019). Multi-hop reading comprehension through question decomposition and rescoring. In *ACL*.

Prior, A. (1968). Past, present and future.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. (2018). Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*.