From Algebraic Word Problem to Program: A Formalized Approach

Wiemerslage, Adam
University of Colorado, Boulder
adam.wiemerslage@colorado.edu

Ahmed, Shafiuddin Rehan
University of Colorado, Boulder
shafiuddin.ahmed@colorado.edu

1 Abstract

In this paper, we propose a pipeline to convert grade school level algebraic word problem into program of a formal language **A-IMP**. Using natural language processing tools, we break the problem into sentence fragments which can then be reduced to functions. The functions are categorized by the head verb of the sentence and its structure, as defined by (Hosseini et al., 2014). We define the function signature and extract its arguments from the text using dependency parsing. We have a working implementation of the entire pipeline which can be found on our github repository.

2 Introduction

Natural language understanding is among the most challenging problems in artificial intelligence. The problem tackles a way of deconstructing a piece of text, usually a sentence or a paragraph, into finer components that gives us some clue about its semantics. The difficulty arises due to the ambiguity inherent to language. To this effect, automatic word problem solving has attracted a lot of research recently (2014, 2014). These problems have a concise structure where sentences can be reduced to equations. While computers are extremely efficient at solving complex problems defined mathematically, the task of understanding word problems as simple as the ones we see in grade school algebra, still poses a great challenge.

A typical algebraic problem begins by describing a partial state of the world as a map of variables. These variables are defined by using the quantifiable entities in the sentence along with its possessor or container. It then continues by adding new variables to the state or updating the existing ones. Towards the end, they generally ask a question to lookup the value of one or more of these variables. Example 1 illustrates a typical kind of problem which follows the above work-flow.

Example 1. *Pooja has 3 apples. She eats one apple. How many apples does she have now?*

In the above example, one would begin by creating a new variable to store the value of the quantifiable object: apples of which the possessor is Pooja. Then they would subtract one from it upon seeing consumption of an apple. Finally, the question asked in the problem would be answered by writing down the current value of that variable. Straightaway, we can identify some of the things the machine might have to identify to generate the work-flow. Things such as the pronoun "She" in the second and third sentences refers to a name Pooja. The term "eats" points to a subtraction event, and many more. In this paper, we present some techniques used to tackle these problems.

Category	Example
Observation	There **were** 28 <u>bales</u> of hay in the *barn*.
Positive	*Joan* **went** to 4 football games this year.
Negative	*John* **lost** 3 of the violet <u>balloons</u>.
Positive Transfer	*Mike's dad* **borrowed** 7 <u>nickels</u> from *Mike*.
Negative Transfer	*Jason* **placed** 131 <u>erasers</u> in the *drawer*.
Construct	*Karen* **added** 1/4 of a cup of <u>walnuts</u> to a *batch* of trail mix.
Destroy	The *rabbits* **ate** 4 of *Dan's* <u>potatoes</u>.

Table 1: Examples for different verb categories in sentences. <u>Entities</u> are underlined; *containers* are italic, and **verbs** are bolded.

3   Previous Work

Automatically solving algebraic word problems is a long-standing AI problem with work dating back to the 60s (N.). There are predominantly two ways of approaching this problem: a) by template matching (Kushman et al., 2014), b) by verb categorization (Hosseini et al., 2014). In the template matching approach, supervised machine learning is used by mapping a large dataset of types of problems labeled with their corresponding equations containing the variables. The variables are then identified from the text by performing yet another classification using the NLP features such as dependency parse, parts of speech, etc. This method, as the authors have shown, works well when solving systems of linear equations.

In the verb categorization approach (focus of our paper), Hosseini et al., try to solve the problem more formally by breaking the problem into fragments and classifying its root verb into operations. They came up with 7 such verb categories that define the type operation a sentence most likely would point to. The categories with examples is illustrated in Table 1 of their paper.

We use a subset of these definitions in our method and attempt to map a sentence to these categories, in a formalized rule-based approach. Once we map them to their "verb signature" we use dependency parsing to extract the arguments from the sentence.

4   Our Method

We implement a method to solve word problems that is similar to Hosseini et al. (2014). Our method assumes a very similar Verb Categorization objective, and attempts to map the relevant arguments of a Verb to the arguments of some function. A large difference is that we attempt to map to a semantic representation, which is really a set of function signatures. We can then perform a deterministic transduction into a formal programming language, as shown in 1, which can be executed to compute some solution.

Because, unlike previous approaches to this task, we do not have labeled training data, we employ an almost purely rule-based approach to this end. This is done under the assumption that word problems for simple algebra have sufficiently similar linguistic structure to make a rule-based approach generalize to some degree. These rules are formed from the natural language processing features that are extracted from a given word problem via out-of-box tools trained on large sets of English data. Simple heuristics combined with some basic unsupervised learning are employed in order to classify verbs into function signatures. We then execute our rule based system to fill the arguments of this function per its type signature. Simple keywords are used finally to identify when a question is being asked, and that the system thus needs to retrieve, or compute, the value for the current state of some variable.
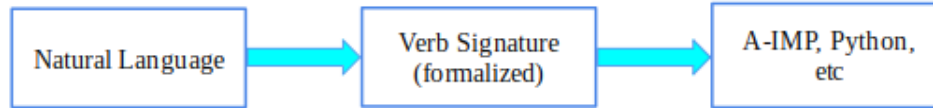
Figure 1: The input language is compiled into a verb signature, which is easy to transduce into a programming language
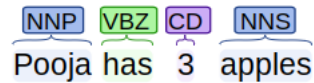

Figure 2: A visual representation of the part of speech tags, produced by Stanford CoreNLP at corenlp.run

5 NLP background

The field of Natural Language Processing provides several methods for representing the structure of natural text. Two extremely import representations are those of part of speech tags, and dependency parses. Part of speech tags are the set of grammatical categories for any word in a language. There are several part of speech taggers implemented in many languages for assigning tags to each word in a sequence, like 'Noun' or 'Verb'. A dependency parse is a representation that categorizes the relationships between words, as in 'Direct Object' or 'Subject'. There are several grammatical theories of parsing, but the dependency representation tends to be favored in computational linguistics for its flat (as opposed to organized into hierarchical categories), graph-like structure.

The dependency parse of a sentence represents each word as a 'dependent' of another word. Intuitively, each word, then has a 'head' to which it is a dependent, except for the root (usually the main verb of a sentence). This syntactic representation of a verb and its dependents is easy to relate to a function and its arguments. We use out-of-box tools for tagging and parsing input text, each of which is contained in the Stanford CoreNLP toolkit Manning et al. (2014).

We also make use of the coreference resolution in that toolkit. Coreference refers to the case of identifying multiple words or expressions that refer to the same thing. As in, 'Pooja took a nap because she was tired.', where we need to represent that 'Pooja' and 'she' are actually referring to the same person. Resolving this issue is very difficult when some reference is not expressed directly, and is 'null-instantiated'. As in, 'Pooja bought a sandwich, and then she ate (the sandwich)', where 'sandwich' is not expressed overtly, but is implied, and is thus not marked as referencing the same thing. This is a difficult problem in NLP and remains unsolved in this work.
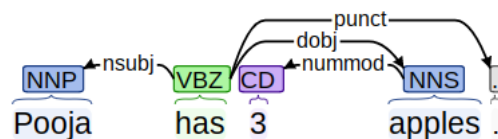

Figure 3: A visual representation of the dependency parse, produced by Stanford CoreNLP at corenlp.run
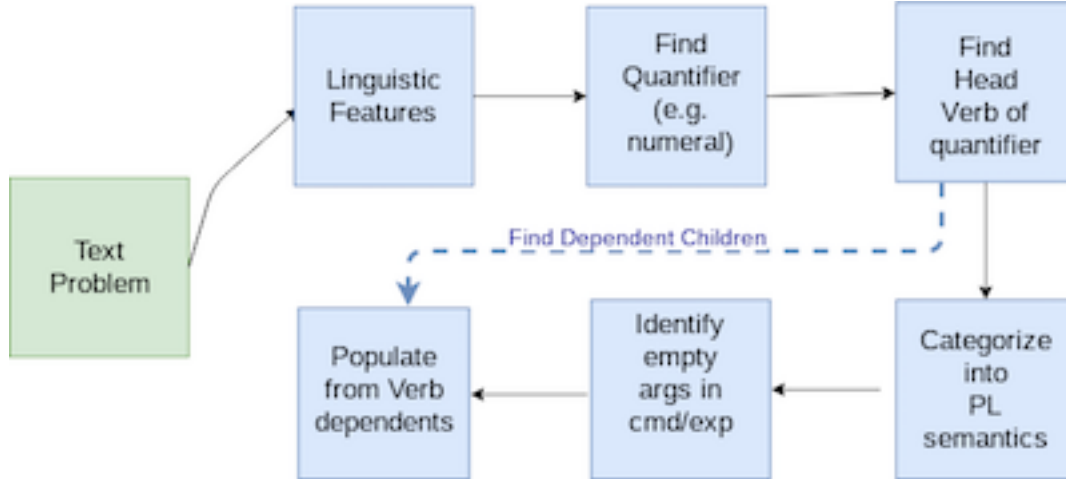
Figure 4: High level overview of the architecture

6 Architecture

6.1 Work-flow

The general work-flow of our system is pictured in figure 4. NLP features are first extracted from the raw text, the structure of which is used to infer the properties needed to emit a verb signature. This verb signature is populated with arguments found in the NLP features, and transduced into **A-IMP**.

6.2 Language: **A-IMP**

$\boxed{e : \tau}$

$$\frac{}{\texttt{addr}[a] : \mathsf{num}} \qquad \frac{}{\texttt{num}[n] : \mathsf{num}} \qquad \frac{e_1 : \mathsf{num} \qquad e_2 : \mathsf{num}}{\texttt{plus}(e_1;e_2) : \mathsf{num}} \qquad \frac{e_1 : \mathsf{num} \quad e_2 : \mathsf{num}}{\texttt{minus}(e_1;e_2) : \mathsf{num}}$$

$\boxed{c : ok}$

$$\frac{e : \mathsf{num}}{\texttt{set}[a](e)\,\mathsf{ok}} \qquad \frac{}{\texttt{skip}\,\mathsf{ok}} \qquad \frac{c_1\,\mathsf{ok} \quad c_2\,\mathsf{ok}}{\texttt{seq}(c_1;c_2)\,\mathsf{ok}} \qquad \frac{e : \mathsf{num}}{\texttt{get}(e)\,\mathsf{ok}}$$
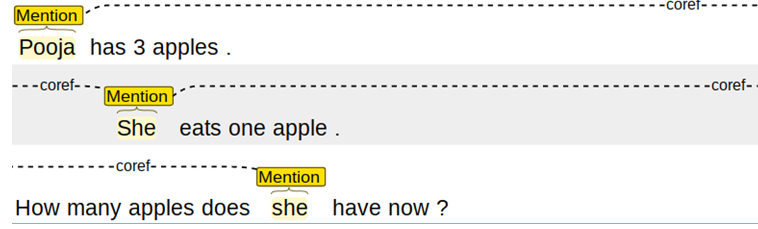
Figure 5: A Co-reference resolution for Example 1 using Stanford CoreNLP

Syntax chart:

Typ	τ	::=	num	num	numbers
			bool	bool	booleans
Exp	e	::=	addr$[a]$	a	addresses (or "assignables")
			num$[n]$	n	numeral
			bool$[b]$	b	boolean
			plus$(e_1;e_2)$	$e_1 + e_2$	addition
			minus$(e_1;e_2)$	$e_1 - e_2$	subtraction
Cmd	c	::=	set$[a](e)$	$a := e$	assignment
			skip	skip	skip
			seq$(c_1;c_2)$	$c_1; c_2$	sequencing
Addr	a	::=	a	a	addresses

Using **A-IMP**, we can inductively define the verb signatures as:

Verb Signature Semantics:

observation$(a;e)$	::=	set$[a](e)$		new observation
construct$(a;e)$	::=	set$[a]($plus$($addr$[a];e))$		addition
destroy$(a;e)$	::=	set$[a]($minus$($addr$[a];e))$		subtraction
ptransfer$(a_1;a_2;e)$	::=	seq$($construct$(a_1;e);$destroy$(a_2;e))$		positive transfer
ntransfer$(a_1;a_2;e)$	::=	seq$($destroy$(a_1;e);$construct$(a_2;e))$		negative transfer
get(a)	::=	get$($addr$[a])$		lookup

6.3 Preprocessing

Our preprocessing involves breaking down the problem into fragments where each of them can be converted into an equation without any ambiguity. For this purpose, the preprocessing involves two steps:

6.3.1 Co-reference Resolution

In the earlier section, we have mentioned what co-reference resolution means. We use the toolkit provided by Stanford to resolve mentions to the same entity in the text. This becomes extremely vital for the variable name assignment and identification in our work-flow. The word or phrase that references a previously introduced entity is replaced with the original text in order to build consistent variable names. In the example, we replace the pronoun "She" with the original mention "Pooja" in the second and third sentence.

6.3.2 Conjunction Breaking

The next step is to break conjunctions in a sentence in such a way that the fragment has all the information required to convert it into an equation. There are multiple forms of conjunctions a sentence can have. The

easiest conjunction is when it separates to multiple root verbs. Take for example the sentence *Pooja has two apples* and *John has one apple.* The conjunction "and" separates the two "has" which have different sets of subject-object pair. And, thus can easily split without losing information.

When a conjunction is present between the subjects or objects of the root verb, we split the sentence by retaining the same root verb in each fragment. Then, the missing subject or object is filled by the corresponding node from the original sentence's verb. For example, a sentence like *Pooja has two apples* and *three oranges* will be split into *Pooja has two apples*, *Pooja has three oranges*. As you may observe, the subject Pooja gets passed onto the second sentence. We found that the Stanford's Enhanced Dependency works extremely well in identifying verbs with multiple subjects and objects (Schuster and Manning, 2016). Conjunctions between adjectives are deleted.

The final output of the preprocessing step is a list of sentences that can be mapped to their verb signatures with the correct arguments.

6.4 Verb Categorization

Our algorithm then begins by identifying a quantifier: some value, or quantifiable word like 'many', or 'much'. We then traverse the dependency graph for the closest parent verb to that quantifier. This verb is the root of a meaningful expression, which will become a verb signature that contributes to some update of our program state. If we have the sentence "Pooja has 3 apples" as in figure 5, then 3 will be identified as a quantifier, and we traverse the graph to the first word tagged as a verb, in this case 'has'.

Next, we can classify that verb into a specific verb signature based on its semantics, the number of arguments that verb signature suggests, and the typing judgments that define it. To do this, a handle on the variables that are being referenced in the dependency graph is needed.

6.4.1 Variable Name Inference

The process of inferring the names of the variables that participate in a verb event are as follow:

1. Get the subject-like dependent of the verb

2. Recursively traverse the dependents of the subject, storing the text of any modifier-like words.

3. Combine the text for the subject and its modifiers, delimiting by an underscore

4. Get the direct-object-like-argument of the verb

5. Apply the same process as step 2 in order to build an underscore delimited string

6. Concatenate the modified subject text, and modified direct object text, delimiting by an underscore

We define subject-like, direct-object-like, and modifier-like as a generalization over groups of tags from the dependency parse. This is to ignore some of the finer grained grammatical details expressed by this parser, and simply capture any information necessary to define variables and a verb signature. In the case of a subject-like argument, its modifier could be a possessor, e.g. *Pooja's Mom*, which would become the variable pooja_mom. In the case of the direct-object-like dependent, it is the same, but more likely to be some nominal or adjectival modifier as in *the green apple*, which would become green_apple.

An assumption being made here is that a verb in this instance will have a subject and an object. Generally in natural language this is not always going to be the case. There are many intransitive verbs that do not take objects, and many subjects that are not overtly expressed. But because we know that the verb is parent to some quantifier, we assume that there is a dependent object being quantified. We also assume that there is an overtly expressed subject due to preprocessing.
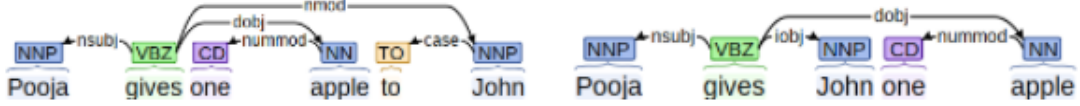
Figure 6: Two examples with indirect-object-like dependents: iobj and nmod, in this case

6.4.2 Heuristic Candidate Selection

We apply simple heuristics on the dependency parse and variable names in order to narrow down a set of at most two candidate verb signatures. Simple matching on question mark (?) symbols can indicate that the word problem is prompting for a solution. Our algorithm looks for a question mark, and an unvalued quantifier, e.g. *many*, or *much*, to select the *get* verb signature.

If this is not the case, then the algorithm checks whether the variable, found beforehand, has already been initialized in our program. To accomplish this we track a list of variable names, and if the variable in the scope of the current sentence does not exist, we select the *observation* verb signature. In the case that the variable is already initialized in our program, the candidate verb signatures are further split into two groups.

The algorithm looks for an indirect-object-like dependent of the head verb, and if it finds one, then it needs to disambiguate between a *positive_transfer* and a *negative_transfer* verb signature. Otherwise, the candidate set is constrained to a *construct* or *destroy* verb signature. This is defined by the verb signature itself, in the number of arguments that it expects.

6.4.3 Semantic Disambiguation

When the candidate verb signatures are found for a particular sentence, the only thing left to do is to disambiguate between a positive_transfer and negative_transfer event, or a construct and destroy event. Intuitively, there is a similarity in both of these distinctions: in both cases the agent of the verb is either giving something away, or getting something. This allows the algorithm to treat the distinction between both cases as the same process of disambiguation. In order to resolve this disambiguation, we experiment with two methods of categorization based on semantic similarity metrics.

In Natural Language Processing, it is common to represent a word as vector in a vector space model. It has been shown that semantic similarity of words can be expressed in the relationship between continuous vectors. These vectors can be found by computing a square matrix M of $\mathbb{R}^{|W|x|W|}$, where W is the words of a corpus. Each $M_{i,j}$ will be the frequency that w_i appears within n words of w_j. Each row of W will thus be rather sparse. A Singular Value Decomposition is then performed over M in order to be able to project each word w_i into a fixed dimensional vector whose values represent the word by the contexts that it appears in (Schütze, 1993). This idea is well founded in the linguistic theory.

Recently, methods for finding fast approximations of these vectors have been developed to great success (Bojanowski et al., 2016), (Pennington et al., 2014). We use word vectors, or 'word embeddings' that have been trained by one such algorithm packaged in a module called Word2Vec (Mikolov et al., 2013), on the massive google news corpus of three billion words. These vectors are freely distributed online. In order to use such vectors for similarity tests, the cosine similarity between any two vectors is computed. In practice, all vectors are normalized to unit length and simply the dot product is used.

$$cosine\ sim = \frac{V_1 \cdot V_2}{||V_1||\ ||V_2||} \tag{1}$$

In one experiment we attempt to make use of VerbNet, a lexical resource that is currently maintained at CU Boulder (Kipper et al., 2008). VerbNet clusters English Verbs into Verb classes: groups of words that behave similarly and have similar meanings. Each class has manually created predicate logic representations

HAS_POSSESSION(START(E), AGENT, THEME) HAS_POSSESSION(END(E), RECIPIENT, THEME) TRANSFER(DURING(E), THEME) CAUSE(AGENT, E)

Figure 7: A VerbNet predicate that expresses a negative_transfer

to express a use of a member of that class. We attempt to get a handle on verb classes via these predicates, and compare the word embedding for the head verb of a given sentence to that of the word embedding for each member verb in a matching class. We categorize VerbNet classes into one of the two categorizations, which can be referred to as positive and negative, based on if they have a predicate that indicates some change of possession or change of location event. In the positive case, entailing positive_transfer and construct, a matching predicate will express that the object is in the location of, or possession of the agent in an end state. The inverse will be true for the negative, entailing negative_transfer and destroy categories, as in figure 7.

Each verb member would be considered to inherit the verb signature category from its parent class, and we simply select the category of the VerbNet member with the highest cosine similarity to embedding for the input verb. Unfortunately, the semantic predicates of VerbNet are in a state of being updated and it proved difficult to make this selection process work correctly due to some inconsistencies in the way that the relevant semantics is currently expressed.

Instead, we manually annotate some verbs that appear in our corpus of algebra problems with a positive or negative class. We then sample 30% of the annotated verbs over an approximately even distribution of positive and negative verbs. The same selection by cosine similarity per word is used. This works fairly well, although not in all cases. One potential issue is that some verbs in opposing classes are antonyms. The word embeddings for antonyms are known often have very close cosine similarity.

Finally, the variables and quantifier that have been identified populate the arguments of the verb signature per its semantics.

7 Conclusion

We introduce a strategy for solving basic algebraic word problems by essentially compiling natural language into a a set of user defined verb signatures. These verb signatures can be defined in the semantics of any arbitrary programming language. When the system classifies a sentence or phrase into a verb signature, that the arguments of that verb signature can be populated based on the specification for its semantics. In this way, the formalization of a programming language allows the mapping of natural language to a series of expressions and commands, and the math that actually produces a solution to be decoupled. This means that the process of extending this system requires only that a new verb signature be defined, and then a mapping from natural language to exactly that verb signature.

Our method relies on a rule based approach that draws upon the fact that basic word problems in algebra have somewhat similar structure. This leads to several linguistic generalizations of syntax that can be applied to many different methods for expressing the same thing in natural language. We hypothesize that this process, and these assumptions can hold for other text problems with a similarly restricted domain.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with sub-word information. *arXiv preprint arXiv:1607.04606*, 2016.

M. J. Hosseini, H. Hajishirzi, O. Etzioni, , and N. Kushman. Learning to solve arithmetic word problems with verb categorization. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533, 2014.

K. Kipper, A. Korhonen, and N. et al. Ryant. "a large-scale classification of english verbs". *Lang Resources & Evaluation*, 2008.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. In *the Annual Meeting of the Association for Computational Linguistics*, 2014.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014. URL `http://www.aclweb.org/anthology/P/P14/P14-5010`.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints*, January 2013.

Shepard Roger N. Edward feigenbaum and julian feldman (editors). computers and thought. new york: Mcgrawhill, 1963. *Behavioral Science*, 9(1):57–65. doi: 10.1002/bs.3830090109. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/bs.3830090109`.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

Sebastian Schuster and Christopher D. Manning. Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *Language Resources and Evaluation (LREC)*, 2016. URL `https://nlp.stanford.edu/pubs/schuster2016enhanced.pdf`.

Hinrich Schütze. Word space. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 895–902, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-274-7. URL `http://dl.acm.org/citation.cfm?id=645753.758140`.