Natural Language Programming with Automatic Code Generation towards Solving Addition-Subtraction Word Problems

Sourav Mandal
Haldia Institute of Technology,
India
sourav.mandal@hithaldia.in

Sudip Kumar Naskar
Jadavpur University,
India
sudip.naskar@cse.jdvu.ac.in

Abstract

Solving mathematical word problems by understanding natural language texts and by representing them in the form of equations to generate the final answers has been gaining importance in recent days. At the same time, automatic code generation from natural language text input (natural language programming) in the field of software engineering and natural language processing (NLP) is drawing the attention of researchers. Representing natural language texts consisting of mathematical or logical information into such programmable event driven scenario to find a conclusion has immense effect in automatic code generation in software engineering, e-learning education, financial report generation, etc. In this paper, we propose a model that extracts relevant information from mathematical word problem (MWP) texts, stores them in predefined templates, models them in object oriented paradigm, and finally map into an object oriented programming (OOP)[1] language (JAVA) automatically to create a complete executable code. The codes are then executed automatically to output the final answer of the MWP. The proposed system can solve addition-subtraction type MWPs and produced an accuracy of 90.48% on a subset of the standard AI2 arithmetic questions[2] dataset.

1 Introduction

Solving MWPs is a very longstanding research problem; researchers in the field of Artificial Intelligence (AI), machine learning and NLP have proposed various methodologies for solving MWPs since 1960s. Word problems are formed using natural language text rather than in mathematical notations (Verschaffel et al., 2000) and they can be of any type of numerical problems based on domains like mathematics, physics, geometry, etc. (Mukherjee and Garain, 2008). Addition-subtraction type MWP is an integral part of basic understanding of mathematics and elementary school level curriculum. The objective of the work presented here is primarily to generate computer programs automatically from natural language texts which when executed will produce the desired answer. Comparison with existing MWP solvers are not appropriate to this kind of work as our end objective is not exactly to create the equation and solve the problem itself, but rather to generate a computer program to solve the problem and thus it adds a new dimension to research in solving MWPs. For example, "*Dan has 64 violet marbles, he gave Mary 14 of the marbles. How many violet marbles does he now have?*" is a word problem which is related to the subtraction or addition operation. This particular problem can be solved manually by noticing the structure of the problem statement in which the first sentence indicates an 'assignment' operation and the second sentence indicates a 'subtraction' operation associated with the verb 'give' for Dan (*primary_owner*) and 'addition' operation for Mary (*secondary_owner*). Final answer requirement is related to violet (*attribute*) marble (*item*) in the possession of Dan. The answer to this problem is simply obtained by subtracting 14 from 64, i.e., 64-14=50.

In the OOP approach, we define 'classes' to represent real life entities and declare instances of those classes called 'objects'. To solve such problems, a computer programmer basically defines a class – 'Person' with the data fields like *name*, *item_name*, *item_attribute* and *item_quantity*, and

[1] http://docs.oracle.com/javase/tutorial/java/concepts/

[2] http://allenai.org/data.html

a method, e.g., *evaluate_result*(). Then he declares objects 'obj1' and 'obj2' of this class. Therefore, for the said example,
$obj1.name = Den$,
$obj1.item\_name = marble$,
$obj1.item\_attribute = violet$ and
$obj1.item\_quantity = 64$
$obj2.name = Mary$,
$obj2.item\_name = marble$,
$obj2.item\_attribute = violet$ and
$obj2.item\_quantity = x$ (not given).
The operation associated with the verb 'has' is '=' (assignment or observation) and can be coded as $obj1.item\_quantity = 64$. The operations associated with the verb 'give' are both '-' and '+' i.e., subtraction and addition (negative_transfer) and can be coded as
$obj1.item\_quantity = obj1.item\_quantity - 14$ and $obj2.item\_quantity = obj2.item\_quantity$ $+14$.
The arithmetic operators are selected based on the verb categories (cf. Table 2) they belong to and the operations can be executed from within a method, e.g., 'evaluate_result()'. In the present work, the verb categorization is rule-based and is determined from the verb predicates (cf. Table 1).

The system first extracts and stores all the required information for the key entities – owners, items, attributes, quantities, and the arithmetic operations relevant to the verb semantics from the MWP text. Then the system creates composite object entities resembling each unique owner-item-attribute combination in the MWP, finds their states and corresponding state transitions (if any) on the basis of the operations or activities (verbs they face) in that MWP, and generates the relevant computer codes. However, automatic extraction of information from natural language text and computer code generation are not trivial. Moreover, solving MWPs requires natural language understanding and reasoning which are very difficult and most of the research in natural language processing (NLP) tend to do away with it. Therefore, solving MWPs automatically has remained an open research challenge.

However, presently our system is unique in three ways. Firstly, our system tries to capture how a programmer can solve an MWP problem using a JAVA like language and it acts as a bridge between unstructured natural language and structured formal language(s). This transformation from natural to formal language (executable program) throws immense challenges in the field of NLP and Information Extraction (IE). Secondly, OOP paradigm is used to model real world data driven tasks and operations. Word problems are apt to be modeled with OOP since it contains real world entities and their specific activities, which motivated us to use an object oriented approach for the present work. The mathematical equation formation is not important here as all operations are represented with JAVA programming statements which determine the mathematical expressions. Once the desirable complete JAVA program (cf. Figure 3) is formed automatically, rest of the activities like compilation and execution of the program to process the result, are handled by the JAVA compiler itself like any computer language programming assignment and here lies the advantage of the proposed approach. Finally, the proposed approach keeps track of all the entities and their state transitions throughout the text (cf. Figure 2) which makes it much easier to answer any question based on the text, not just the question actually present in the MWP problem. It does not have to start processing afresh for answering any other question based on the same text.

The remainder of the paper is organized as follows. Section 2 presents an overview of relevant related work. Section 3 provides a detailed discussion on the system components. Section 4 outlines the datasets, experiments and the corresponding results together with some analysis, followed by conclusions and avenues for further research in Section 5.

2 Related Work

The research problem on generation of executable computer programs for solving MWPs has not been attempted so far to the best of our knowledge. However, formal language modeling from natural language text has been studied previously in various domain by researchers mainly in software engineering (Bryant et al., 2003; Lei et al., 2013), web interfaces of databases (Alexander et al., 2013), etc. Some researchers tried to represent natural language texts using regular expressions (Kushman and Barzilay, 2013). Ballard and Biermann (Ballard and Biermann, 1979) proposed a natural language computing ('NLC') prototype to process and evaluate small natural language text word problems based on matrix computation. They proposed a method to generate solution from a matrix entry and solve problems like "*add five with the second positive entry in row 5*", "*double the fifth entry and add that to the last entry of that row*", etc. Each of these assignments have some types of mathematical terminologies like 'add', 'double', etc., which clearly indicates the operation or operator. This research problem is not exactly related to automatic program generation, rather it is about processing a matrix data structure syntactically to generate the desired result based on matrix arithmetic. Liu and Lieberman (Liu and Lieberman, 2005) developed a system 'Metafor' which converts a small description of an event into a 'Python' program based on interaction logs with respect to time and entity participation. Kate et al. (Kate et al., 2005) tried to represent natural language texts syntactically and semantically into a formal representation that is based mainly on deterministic context-free grammar. They used "if-then" rules to develop a new formal language 'CLANG' for processing natural language text. Mihalcea et al. (Mihalcea et al., 2006) first proposed a system that attempts to convert natural language texts directly into computer programs. They tried to identify various algorithmic steps, decisions and loop structures from English text representing any event and convert it into a program skeleton using 'PERL' programming language which is object oriented in nature. Following the "who_does_what" structure their system develops a program skeleton and generates the 'PERL' code for texts like "*When customer orders a drink, the bartender makes it*". They developed a model which creates different classes like 'Customer', 'Bartender' and relevant methods like 'order_drink()', 'make_drink()' to support their actions. Our work is little relevant to their work. Alongside, many researchers proposed various methodologies to solve MWPs (Kushman et al., 2014; Hosseini et al., 2014; Walker and Kintsch, 1985; Fletcher, 1985; Roy and Roth, 2015; Shi et al., 2015; Mitra and Baral, 2016). The work presented in this paper differs from these works.

3 System Description

3.1 Mapping Input Texts to The Concept

Natural language texts representing some MWPs typically contain multiple factual sentences and a 'question sentence' at the end (cf. the example given in Section 1). Each sentence may or may not have some mathematical meaning. Our objective is to identify the key players or entities and their state transitions from the first sentence they occur in and till the last sentence. An MWP example containing multiple sentences is given below.

> *"Harry has 15 blue and 10 green balloons. He lost 5 blue balloons in the market. Then he bought 3 green balloons from a shop. Tim has 12 kites and 10 blue balloons. Tim gave Harry 4 blue balloons.. ... How many green balloons does Harry have? "*

This MWP problem involves 2 persons having 2 types of balloons, blue and green, and 1 person having kites. Here owner entity names are 'Harry', 'Tim', and item entity names are 'balloon' and 'kite', and item attributes associated with the item 'balloon' are 'blue' and 'green'. Our objective is to map such information expressed in natural language texts into object oriented programming paradigm. Every sentence is considered as a state and throughout the input text several state transitions take place with all unique 'Owner–Item–Attribute'(OIA) objects (cf. Figure 2). Here we create objects like 'Harry-balloon-blue', 'Harry-balloon-green', 'Tim-balloon-blue', 'Tim-kite-null' along with their respective quantities. It is to be noticed that the owner does not have to be a person always. Our system identifies all different types of owner, item (and attribute, if any) combinations from the input text and create 'objects' for each of them. It also identifies their state transitions that they go through throughout the problem text. Most importantly, if the text has question sentence like *"How many blue balloons are now with Harry?"* or *" How many kites does Tim have now?"*, the system formulates the answer by matching the 'OIA' object in the question sentence. Our system carries the information about all the 'OIA' objects and the changes in quantities of the items (if any) occurring in association with the operations (related to the 'verbs') they face till their final state. Therefore, after processing the question sentence and identifying the 'OIA' object associated with it, the system displays the final processed quantity of the corresponding 'OIA' object as the answer.

3.2 MWP Text Simplification

To make the processing more convenient, the input text is simplified first. Conjunctions are removed and coreferences are substituted to convert the input text into a simplified format so that we can extract information with out any ambiguities. We use Stanford CoreNLP[3] suite 3.6.0 to perform the intermediate NLP tasks, e.g., POS tagging, dependency parsing, coreference resolution, etc., and extract relevant information. We remove conjunctions like 'and', ',' (comma), 'but', ', and' and ', but' from compound sentences and break them into multiple simple sentences. The coreference mentions for pronouns like 'he', 'she', 'his', 'her' etc., are substituted with the corresponding referred expressions so that we can extract the owner entities directly and unambiguously.

3.3 Information Extraction based on Semantic Role Labelling (SRL)

SRL techniques are mainly used to semantically process texts and to define role(s) of every words present in a text. For extracting information from text, we used the SRL tool – Mateplus[4] (Roth and Woodsend, 2014; Roth and Lapata, 2015), which was developed for meaning representations based on the CMU SEMAFOR[5] tool and frameNet[6]. Table 1 shows the output of 'Mateplus' for the sample sentence "*Sam gave Mary 23 green marbles.*". Depending on the type of the predicates and also

ID	Form	POS	Dependency	Predicate	Args:Locating
1	Sam	NNP	SUB	-	Donor
2	gave	VBD	ROOT	Giving	-
3	Mary	NNP	OBJ	-	Recipient
4	23	CD	NMOD	-	-
5	green	JJ	AMOD	-	-
6	marbles	NNS	OBJ	-	Theme
7	'.'	'.'	P	-	-

Table 1: A sample SRL output

the verb grouping from VerbNet[7], the verbs are manually categorized and respective equations are generated by the system (cf. Subsection 3.4). Given the example, the predicate (e.g., 'Giving'), owners (e.g., 'Donor' as primary and 'Recipient' as secondary owner), items (e.g., 'Theme') and the attribute(s) of the item(s) are extracted from the SRL output (cf. 'give', 'Sam', 'Mary','marble' and 'green' in Table 1 respectively). Depending on the type (i.e., category) of the predicates, respective 'operations' are identified for each 'OIA' triplet/object (cf. Section 3.4).

The system extracts all relevant information from the input MWP texts, sentence by sentence, identifying the owners, items, item attributes, 'verb', 'cardinal number' (or 'quantity') from the SRL output (cf. Table 1) using a rule-based approach. These information are extracted from the POS tag, and dependency relations combined with 'predicates' and relevant 'arguments'. For example, *NNP/NN* and *SUB* is an 'owner' entity, *NNP/NN* and *NMOD/PMOD/OBJ* is a 'secondary owner', *NNS/NN* and *OBJ* is an 'item', *JJ* and *AMOD/NMOD* is an 'item-attribute'. A maximum of 5 conditions (rules) are used to identify each type.

3.4 Verb Categorization & Equation Formation

We studied the verbs appearing in the dataset (cf. Section 4) and by manually analyzing the predicates and arguments (cf. Subsection 3.3 and Table 1), we grouped the verbs into 5 categories based on the frameNet frame definitions along with the similarity of the verbs in terms of VerbNet verb grouping and probable arithmetic operational connotation (=, +, -) as in Table 2. We carried out verb categorization motivated by the work of (Hosseini et al., 2014).

Category	**Verbs**	**Operator**
Observation	$have, find$	$assignment$
Increment	$gather, grow$	$+$
Decrement	$lose, spend$	$-$
Positive Transfer	$take, receive$	$+and-$
Negative Transfer	$give, sell, pay$	$-and+$

Table 2: Schema and operations for the verb categories.

For example, using the frame definition of 'Giving[8]' in the 'frameNet', we categorized 'give' in the 'negative transfer' category where '-' operator is associated with the donor/primary_owner (*Sam* in Table 1) and '+' operator is associated with the recipient/secondary_owner (*Mary* in Table 1). The frame definition for 'Giving' is (*Donor, [Recipient], Theme/Items, [Quantities], [Time], [Location]....*). Similarly, we categorized

[3]http://stanfordnlp.github.io/CoreNLP/

[4]https://github.com/microth/mateplus

[5]www.ark.cs.cmu.edu/SEMAFOR

[6]https://framenet.icsi.berkeley.edu

[7]http://verbs.colorado.edu/~mpalmer/projects/verbnet.html

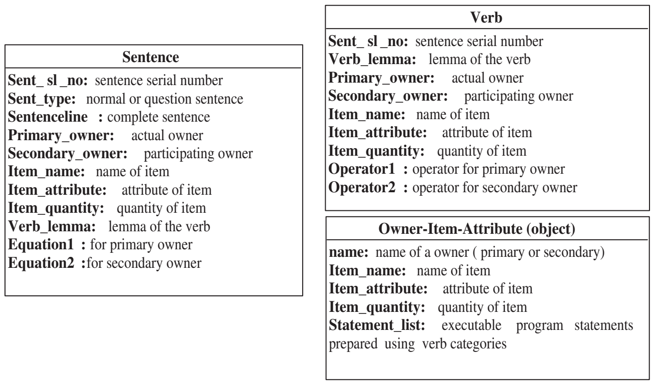
[8]https://framenet2.icsi.berkeley.edu/fnReports/data/frameIndex.xml?frame=Giving

Category	Examples	Schema Entry	Equations
(Null) Observation	Joan has 40 blue balloons	[Joan, null, balloon blue, 40]	Joan-balloon-blue.quantity=40
Increment	Tom grew 9 watermelons	[Tom, null, watermelon, null, 9]	Tom-watermelon-null.quantity=Tom-watermelon-null.quantity+9
Decrement	Sally lost 2 of the orange balloons	[Sally, null, balloons, orange, 2]	Sally-balloon-orange.quantity= Sally-balloon-orange.quantity-2
Positive Transfer	Dan took 22 pencils from the drawer	[Dan, drawer, pencils, null, 22]	Dan-pencil-null.quantity=Dan-pencil-null.quantity+22 and drawer-pencil-null.quantity=drawer-pencil-null.quantity-22
Negative Transfer	Jason gave 13 of the seashells to Tim	[Jason, Tim, seashell, null, 13]	Jason-seashell-null.quantity=Jason-seashell-null.quantity-13 and Tim-seashell-null.quantity=Tim-seashell-null.quantity+13

Table 3: Equation formation based on verb category and schema information

the verbs like 'has', 'find', 'are' having similar kind of frame definitions in the 'observation' category representing the '=' operation as they do not refer any changes. We developed a database schema to store the extracted information from each input sentence by analyzing the predicates associated with the verbs contained in the MWPs. The schema is generic and defined as *[primary_owner, secondary_owner, item_name, item_attribute, item_count]* for all categories of verbs that could be present in the input text sentences. Table 2 presents the verb categories (based on only one sense of the verbs) and the corresponding related operations. The 'Positive Transfer' and 'Negative Transfer' categories represent two operators connected with the 'primary_owner' and 'secondary_owner'.

Table 3 presents the targeted equations related to the verb categories. The schema entry (cf. Table 3) includes extracted information for primary_owner, secondary_owner, item_name, item_attribute, item_quantity from each sentence in the input MWP text. Owner_name (primary_owner or secondary_owner), item_name and item_attribute, these 3 components create a single 'OIA' entity throughout the input text processing. E.g., the sentence "*Jason gave 13 of the seashells to Tim"* contains the primary_owner 'Jason', secondary_owner 'Tim', item_name 'seashell' and item_attribute 'null (no attribute)' (cf. Table 3). Here, 'Jason-seashell-null' and 'Tim-seashell-null' can be referred as two 'objects', say 'Object[0]' and 'Object[1]', in the OOP scenario where the 'item_quantity(i.e. 13) is subtracted (i.e., -) from 'Object[0]' and added to 'Object[1]' since the verb 'give' belongs to the 'Negative Transfer' category. Similarly, the sentence "*Joan has 40 blue balloons*" will create an object entity 'Joan-balloon-blue' where the 'OIA' object is associated with the 'assignment' ('=') operator with the 'item_quantity'(i.e. 40). The equation formations also follow the same directions as in Table 3. An input text sentence may not always have 'secondary_owner' or 'item_attribute'. Therefore, some of the schema entries are shown as 'null' in Table 3.



Sentence
Sent_ sl _no: sentence serial number
Sent_type: normal or question sentence
Senteneceline : complete sentence
Primary_owner: actual owner
Secondary_owner: participating owner
Item_name: name of item
Item_attribute: attribute of item
Item_quantity: quantity of item
Verb_lemma: lemma of the verb
Equation1 : for primary owner
Equation2 :for secondary owner

Verb
Sent_ sl _no: sentence serial number
Verb_lemma: lemma of the verb
Primary_owner: actual owner
Secondary_owner: participating owner
Item_name: name of item
Item_attribute: attribute of item
Item_quantity: quantity of item
Operator1 : operator for primary owner
Operator2 : operator for secondary owner

Owner-Item-Attribute (object)
name: name of a owner (primary or secondary)
Item_name: name of item
Item_attribute: attribute of item
Item_quantity: quantity of item
Statement_list: executable program statements prepared using verb categories

Figure 1: Template-based information extraction.

3.5 Information Processing & Template Filling

We followed a template based IE approach and used three templates – 'Sentence', 'OIA' and 'Verb'. Figure 1 describes the three templates. After extracting all the relevant information, the system stores them in the 'Sentence' and 'Verb' templates. Successively, the system identifies each unique 'OIA' triplet such that at least one component is unique with respect to the other triplets (cf. Section 3.6). This procedure creates a number of instances of 'OIA' template based on the identified unique triplets. Then by processing the extracted information in the 'Verb' template, the desired equation(s) is generated with the associated 'OIA' triplet(s) according to the verb category (cf. 'Equation' column in Table 3) for each sentence. The generated equations are then added in the 'sentence' template as 'Equation1' (for primary owner) and as 'Equation2' (for secondary_owner) (cf. Figure 1). Finally, real programming 'objects' are created related to all the 'OIA' triplets. By matching and replacing the

Owner-Item-Attribute / Objects	Item_ count(x)	Verb_ lemma	Operation	Equation statements	State no. / Sentence no.
mike-balloon-orange / obj[0]	8	have	assignment	obj[0].quantity=8	1 / 1
Sam-balloon-orange / obj[1]	14	have	assignment	obj[1].quantity=14	1 / 2
mike-balloon-orange / obj[0]	4	give	-	obj[0].quantity= obj[0].quantity-4	2 / 3
Sam-balloon-orange / obj[1]	4	give	+	obj[1].quantity= obj[1].quantity+4	2 / 3

Table 4: Generating program statements

triplets with the respective actual objects in the 'equations' (cf. 'Equations' column in Table 3) of the 'Sentence' templates, the actual JAVA programming statements are created (cf. Subsection 3.6 and Table 4). These program statements are then appended according to the sequence of occurrence of the 'OIA' objects in a MWP following their state diagrams (cf. Figure 2), which make up the executable JAVA program (cf. Figure 3). The system generates 'Sentence' templates equal to the number of sentences in an MWP, 'Verb' templates equaling the numbers of verbs existing, and an 'OIA' template for each unique 'OIA' object. The last 'Sentence' template is the 'question sentence' which is separately analyzed to identify question requirements. Extracted and processed information stored in templates are finally stored in tables using a relational database approach – MYSQL[9]. This introduces structure into the unstructured natural language texts and also makes the processing and reasoning tasks easier.

3.6 Automatic Program Generation Using OOP Approach

3.6.1 Object Creation

In order to generate an object oriented program from the input text, the first task is to represent the identified unique 'OIA' combinations as real 'Objects' in OOP. E.g., if after simplification, the input MWP text is "*Mike has 8 orange marbles. Sam has 14 orange marbles. Mike gave Sam 4 of the marbles. How many orange marbles does Mike now have?*", the system identifies 2 unique 'OIA' combinations – 'Mike-marble-orange' and 'Sam-marble-orange'. The identified 'OIA' triplets are then represented as 'obj[0]' and 'obj[1]' using the OOP concept. These objects are the real instantiation of the predefined class 'OwnerItem' (cf. Figure 3) resembling an 'OIA' template. The system dose not consider the question sentence for object creation.

3.6.2 JAVA Program Statements Generation

For the example in Section 3.6.1, the verb, 'have', belongs to the observation category (cf. Table 2) and therefore generate the assignment ('=") statements. The verb 'give' is the type of 'negative transfer' (cf. Table 2)and it produces the statement having subtraction operation ('-') with the primary_owner 'Mike' (obj[0]) and addition operation ('+') with the secondary_owner 'Sam' (obj[1]), shown in the 'Equation statements' column in Table 4. The statements as a whole lead to the executable program statements in JAVA language. Table 4 shows, how the 'OIA' objects are created, corresponding values are associated to the objects, the mathematical operations are identified from the verb lemma and the corresponding program statements are generated from the same example. The equations are first formed, e.g., "*mike-balloon-orange.quantity=8*" (like 'Equations' column in Table 3) in which the 'OIA' objects are later replaced by real objects and new equations are formed, e.g., "*obj[0].quantity=8*" (cf. 'Equation statements' column in Table 4) using JAVA programming syntaxes.

3.6.3 State Transition Diagram

Figure 2 demonstrates a simple forward "state transition diagram" for all 'OIA' combinations or resultant 'OwnerItem' objects for the example mentioned earlier. A 'state' of an 'OwnerItem' objects is basically the sentence in the MWP texts where it exists. An 'OwnerItem' object in the program appears first in any one of the sentences

[9]`http://www.mysql.co\ref{tab:VerbCategories}.m/`

(first state) and moves towards the last sentence they appear in (last state)of a MWP except the question sentence (referred as forward transition). The question sentence does not result in any state change. Generally it does not have any operation associated with it. Therefore, individual ‘Owner-

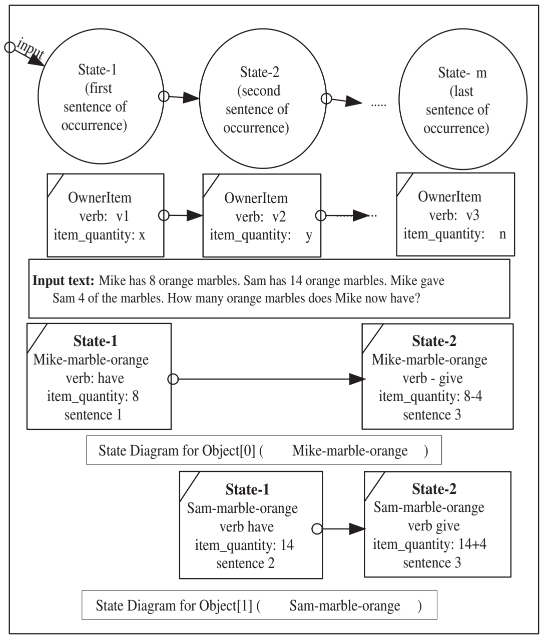
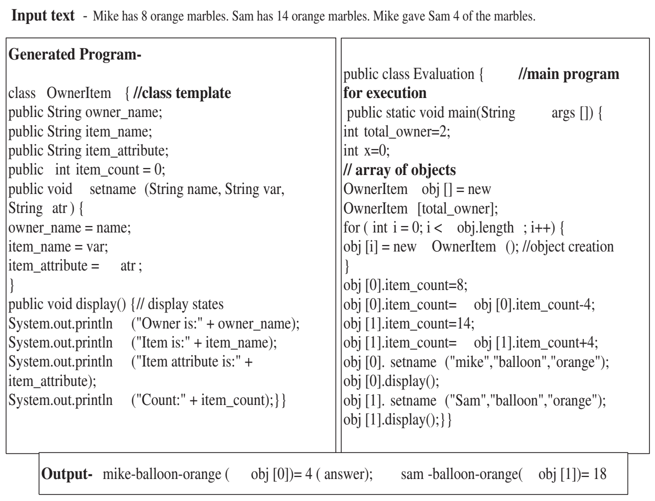

input
State-1 (first sentence of occurrence)
State-2 (second sentence of occurrence)
.....
State- m (last sentence of occurrence)
OwnerItem verb: v1 item_quantity: x
OwnerItem verb: v2 item_quantity: y
OwnerItem verb: v3 item_quantity: n
Input text: Mike has 8 orange marbles. Sam has 14 orange marbles. Mike gave Sam 4 of the marbles. How many orange marbles does Mike now have?
State-1 Mike-marble-orange verb: have item_quantity: 8 sentence 1
State-2 Mike-marble-orange verb - give item_quantity: 8-4 sentence 3
State Diagram for Object[0] (Mike-marble-orange)
State-1 Sam-marble-orange verb have item_quantity: 14 sentence 2
State-2 Sam-marble-orange verb give item_quantity: 14+4 sentence 3
State Diagram for Object[1] (Sam-marble-orange)

Figure 2: State diagrams of the ‘OwnerItem’ objects

Item’ object entities have their own state transition diagrams based on their presence in the sentences. In every sentential state they may participate in an operation or not. In Table 4, the ‘objects’ obj[0] has 2 states, occurring in sentence numbers 1 and 3 and obj[1] has 2 states occurring in sentence numbers are 2 and 3. Figure 2 gives a pictorial representation of the sentential states of the object entities and their forward transitions based on the operations they performed. The last state of any objects are having the final quantity associated to them. Analyzing the extracted information from question sentence the object is identified for whom the answer will be displayed (obj[0] in the Figure 2). The state diagram in Figure 2 is related to the CHANGE type word problems (Mitra and Baral, 2016) having all quantities available for the desired object in terms of answer generation. In some cases, where the problems have an adverbial modifier like all, total, together, etc (COMBINE type (Mitra and Baral, 2016)), it is observed that each unique object has single state and no transition. In such scenario, the statements (or related state quantities) of all relevant objects are summed up to generate complete JAVA code which produces final answer.

3.6.4 Executable Program Generation

After creation of the program statements for all individual ‘OIA’ objects, they are integrated in a predefined JAVA program skeleton in a rule-based manner. The desired program statements are only considered and added according to the sequence of occurrence (i.e., events) in the given MWP. Figure 3 shows the generated program for the same example text. The system processes the question sentence to extract information about the ‘OwnerItem’ about whom (or which) the question has been asked and the presence of any modifier like ‘all’, ‘total’ etc (indicates summation). Subsequently, the extracted information is used to generate additional program statements (to be appended at the end and not given in Figure 3) that processes and displays the desired final answer. After the program generation, compilation and execution of that program are performed by the JAVA compiler (JVM) itself to generate the final answer.



Input text - Mike has 8 orange marbles. Sam has 14 orange marbles. Mike gave Sam 4 of the marbles.

Generated Program-

```
class   OwnerItem   { //class template
public String owner_name;
public String item_name;
public String item_attribute;
public   int item_count = 0;
public void    setname  (String name, String var,
String  atr ) {
owner_name = name;
item_name = var;
item_attribute =      atr ;
}
public void display() {// display states
System.out.println    ("Owner is:" + owner_name);
System.out.println    ("Item is:" + item_name);
System.out.println    ("Item attribute is:" +
item_attribute);
System.out.println    ("Count:" + item_count);}}
```

```
public class Evaluation {        //main program
for execution
 public static void main(String        args []) {
int total_owner=2;
int x=0;
// array of objects
OwnerItem   obj [] = new
OwnerItem  [total_owner];
for ( int i = 0; i <   obj.length  ; i++) {
obj [i] = new    OwnerItem  (); //object creation
}
obj [0].item_count=8;
obj [0].item_count=    obj [0].item_count-4;
obj [1].item_count=14;
obj [1].item_count=    obj [1].item_count+4;
obj [0]. setname  ("mike","balloon","orange");
obj [0].display();
obj [1]. setname  ("Sam","balloon","orange");
obj [1].display();}}
```

Output- mike-balloon-orange (obj [0])= 4 (answer);   sam -balloon-orange(obj [1])= 18

Figure 3: Automatically Generated Program

4 Dataset, Results and Discussions

There is a broad sense of natural language programming available in literature, however, they do not exactly relate to our objective or methodology. Though no standard datasets are available specifically for such work, we compiled a dataset containing 189 questions. We selected word problems from the dateset available with the work of (Hosseini et al., 2014) which is the same as the ‘AI2 Arithmetic Questions’ dataset. They compiled 395 addition-subtraction word problems with 3 subsets

– MA1, IXL, and MA2 with varying degree of complexity. Our selection was based on the constraint that the sentences of each word problems must have links between them towards the forward movement of state transitions and each sentence in a MWP (i) must not have any "missing information" and (ii) must not be an "irrelevant sentence" with respect to answer generation. For example, the problems "*Joan found 70 seashells on the beach. She gave Sam some of her seashells. She has 27 seashells. How many seashells did she give to Sam?*" contains a sentence having the word 'some' which does not hold any definitive cardinal value, instead indicates a operation, are referred to as 'missing information'. Another example from the dataset is "*Tom purchased a Batman game for $ 13.60 , and a Superman game for $ 5.06. Tom already owns 2 games. How much did Tom spend on video games?*. In this example, the sentence "*Tom already owns 2 games.*" does not have any actual relation with the desired result and this kind of sentences are referred to as 'irrelevant sentence'. These cases were not included in the dataset since our system presently does not have the capabilities to handle them. We selected in total 189 problems[10] from MA1 and MA2 (out of total 255 problems) based on the constraints. We did not consider IXL since the corresponding problems involve more information gaps which call for complex reasoning (due to ambiguities in owners, items) that can not be handled by the proposed approach and some problems of MA1 or MA2 do not fit with our objective.

The system generated syntactically correct programs in all cases, however, in terms of correct answer generation (i.e., logically correct programs) it produced an accuracy of 90.48% (171 out of 189) on the test dataset. The system performed properly for texts containing CHANGE or COMBINE information. Cases for which the system did not produce correct results are given below with some examples.

- **No Link Among Owners, Participating Operation:** E.g., Dan had 7 potatoes and 4 cantaloupes in the garden. The rabbits ate 4 of the potatoes. How many potatoes does Dan now have? (8 problems/44.5%)

- **Wrong Program/ Answer Generation Due to Various Reasons like Program Logical Errors, Sentence Sequence, Wrong IE/ SRL etc.:** E.g., There are 7 crayons in the drawer and 6 crayons on the desk . Sam placed 4 crayons and 8 scissors on the desk . How many crayons are now there in total ? (7 problems/38.9%)

- **Improper Reasoning of Question Sentence:** E.g., A restaurant served 9 hot dogs during lunch and 2 during dinner today. It served 5 of them yesterday. How many hot dogs were served today? (3 problems/16.6%)

5 Conclusions

Object oriented analysis and design approach is very useful for modeling any real world data and event-driven scenario with ease. We only need to identify the key entities and their roles in that scenario. The main objective of our work is the generation of structured programs (JAVA based) automatically from natural language MWP texts, not exactly the solution of the MWPs itself, which can be further extended to become a complete MWP solver. The work is more relevant to natural language programming (like Mihalcea et al. (2006)) rather than the development of an MWP solver. We selected the MWP domain since it is event-driven and involves operations like assignment, addition, subtraction, etc., related to the associated verbs. We tested our system on typically small input texts containing only 3–4 sentences (i.e., before text simplification), however, the approach is generic and it will also work for longer input texts. The approach can also be extended for potential use in question answering and summarization purposes by identifying the key players like owners and items for the domains that handle operations like additions and subtractions. If we augment the model with various 'OIA' entities and large number of functionalities then the methodology can represent any natural language text specific to some domain into an object-oriented paradigm and can add great power to automatic code generation from software requirement specifications and software designs. We would also like to extend the proposed OOP based approach to model and solve word problems involving multiplication and division and try to minimize the constraints.

[10]dataset available at: `https://sites.google.com/site/autocodegeneration/`

Acknowledgments

Sudip Kumar Naskar is supported by Media Lab Asia, MeitY, Government of India, under the Young Faculty Research Fellowship of the Visvesvaraya PhD Scheme for Electronics & IT.

References

Rukshan Alexander, Prashanthi Rukshan, and Sinnathamby Mahesan. 2013. Natural language web interface for database (NLWIDB). *CoRR*, abs/1308.3830.

Bruce W. Ballard and Alan W. Biermann. 1979. Programming in natural language: "NLC"; as a prototype. In *Proceedings of the 1979 Annual Conference*, ACM '79, pages 228–237, New York, NY, USA. ACM.

Barrett R Bryant, Beurn-Seuk Lee, Fei Cao, Wei Zhao, and Jeffrey G Gray. 2003. From natural language requirements to executable models of software components. Technical report, DTIC Document.

Charles R Fletcher. 1985. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*, 17(5):565–571.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 523–533.

Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 1062–1068.

Nate Kushman and Regina Barzilay. 2013. Using semantic unification to generate regular expressions from natural language. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 826–836.

Nate Kushman, Luke Zettlemoyer, Regina Barzilay, and Yoav Artzi. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 271–281.

Tao Lei, Fan Long, Regina Barzilay, and Martin C. Rinard. 2013. From natural language specifications to program input parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 1294–1303.

Hugo Liu and Henry Lieberman. 2005. Metafor: visualizing stories as code. In *Proceedings of the 2005 International Conference on Intelligent User Interfaces, January 10-13, 2005, San Diego, California, USA*, pages 305–307.

Rada Mihalcea, Hugo Liu, and Henry Lieberman. 2006. NLP (natural language processing) for NLP (natural language programming). In *Computational Linguistics and Intelligent Text Processing, 7th International Conference, CICLing 2006, Mexico City, Mexico, February 19-25, 2006, Proceedings*, pages 319–330.

Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Anirban Mukherjee and Utpal Garain. 2008. A review of methods for automatic understanding of natural language mathematical problems. *Artif. Intell. Rev.*, 29(2):93–122.

Michael Roth and Mirella Lapata. 2015. Context-aware frame-semantic role labeling. *TACL*, 3:449–460.

Michael Roth and Kristian Woodsend. 2014. Composition of word representations improves semantic role labelling. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 407–413.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1743–1752.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1132–1142.

Lieven Verschaffel, Brian Greer, and Erik De Corte. 2000. *Making sense of word problems*. Lisse Swets and Zeitlinger.

William H. Walker and Walter Kintsch. 1985. Automatic and strategic aspects of knowledge retrieval. *Cognitive Science*, 9(2):261–283.