(1) <u>**Learning to Solve Arithmetic Word Problems with Verb Categorization:**</u>

This paper investigates the task of learning to solve such problems by mapping the verbs in the problem text into categories that describe their impact on the world state. They name the system ARIS.
Attempt to solve the add subtract type problem.

Entities are the objects which are being incremented and decremented. Containers posses the entities. Quantities are assigned to entities possessed by the container. Quantities can be known number or in the form of variable.

A problem text is split into fragments where each fragment corresponds to an observation or an update of the quantity of an entity in one or two containers. Approach is to first categorise verbs of the problem into predefined 7 categories based on their effect on quantities of entities which containers have. The verb in each sentence is associated with one or two containers, and ARIS has to classify each verb in a sentence into one of seven categories that describe the impact of the verb on the containers.

World states are defined for the problem statement. A state is defined as a tuple.
<E, C, R>. Where E is the set of entities, C is the set of containers, and relations R among entities, containers, attributes, and quantities.

<u>State Transitions</u> → Word state is changed by the verb contained in each fragment.
Based on the verb category present in the fragment, the quantities of containers in the entities are updated. Each entity can be associated with one or two containers.

<u>Pipeline of the problem solving</u> → Identifying the noun associated with numbers. → Identifying the verbs involved in the each sentence → verb categorisation → forming the mathematical equation and solving the problem.

<u>Verb Categorisation training</u> → To identify the type of verb involved in the sentence, they train using the binary vector of the category type and features based on different level of problem hierarchy.

<u>Features:</u>

1) <u>Similarity based features</u> → For the verb in the sentence, the similarity between the the verb and the seed verbs(most popular verbs found using L1 regularisation) is found.
2) <u>Word-Net based features</u> → Using the scores for 15 different categories provided by word-nets.
3) <u>Structural Features</u> → 35 different features based on dependencies between the verb and other elements.

Based on the above defined features, Verbs are classified using SVMs.

(2) **Learning to Automatically Solve Algebra Word Problems-**

System which creates reasoning across the sentence boundary to create and solve the system of linear equations. The learning algorithm uses varied supervision, including either full equations or just the final answers.

Problem is solved by first converting the word problem into linear equation and then solving it.

Equation Templates - They decided some of the predefined templates in which a problem statement may occur as linear equation. A template subsequently contains slots of different types viz, noun, numbers. These slots are then mapped according to the word problem.

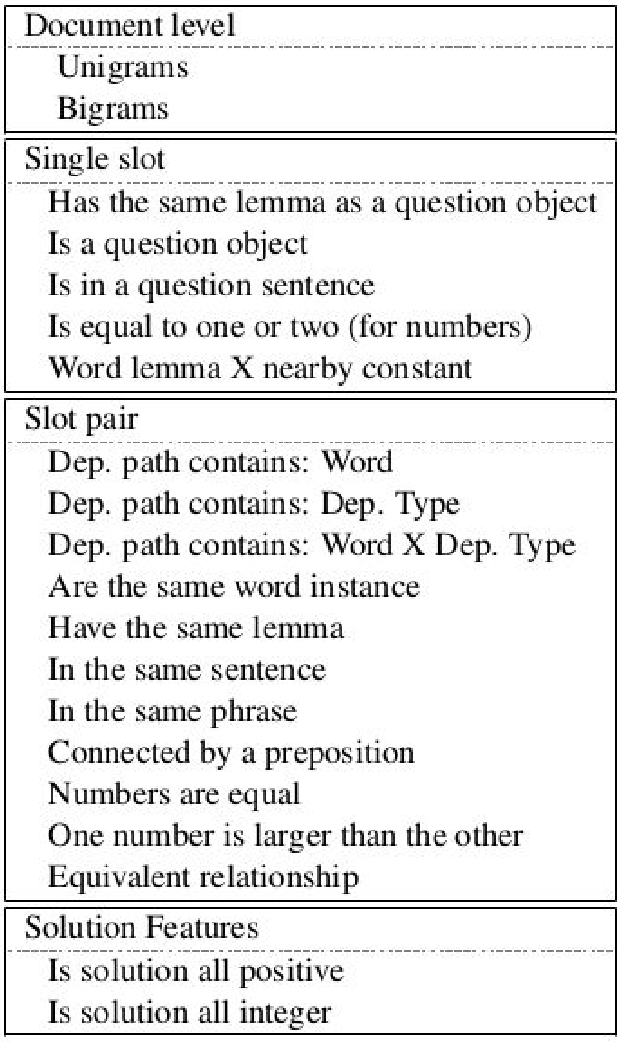
Mapping - Assuming a particular equation template fits for the word problem, system tries to fit in the corresponding nouns, numbers into the slots defined in the template.

Template creation → Templates are created by analysing the examples in the training set.
(a) replacing each variable with an unknown slot.
(b) re-placing each number mentioned in the text with a number slot.

Learning→ Giving a word problem, system needs to learn, which template is best suited for the given problem. Probabilistic methods are used to so the template assignment. Now the models are to be learnt. System estimate the parameters by maximizing the conditional log-likelihood of the data, marginalizing over all valid derivations.

We use L-BFGS (Nocedal and Wright, 2006) to optimize the parameters.
Model Details →
Template Canonicalization: Fuse the equations which are same semantically but have different syntactic representation .
Eg : $j = b+3$ and $j -3 = b$ represent the same thing.
Slot Signature: A noun can be used in multiple slots of the template. There ids are given to the slots.

Features for training the model → System uses several natural language based features to learn the template mappings.



Document level
Unigrams
Bigrams
Single slot
Has the same lemma as a question object
Is a question object
Is in a question sentence
Is equal to one or two (for numbers)
Word lemma X nearby constant
Slot pair
Dep. path contains: Word
Dep. path contains: Dep. Type
Dep. path contains: Word X Dep. Type
Are the same word instance
Have the same lemma
In the same sentence
In the same phrase
Connected by a preposition
Numbers are equal
One number is larger than the other
Equivalent relationship
Solution Features
Is solution all positive
Is solution all integer

L2 regularisation used along with the L-BFGS model.

(3) Automatically Solving Number Word Problems by Semantic Parsing and Reasoning

The aim of the approach is to make a parse tree for a word problem and solve it to get to the answer. They do it by Semantic Parsing of the word problem. Parser is based on context free grammar. Grammar rules for the parser are being learned semi-automatically. They name the system as **SigmaDolphin**.

Approach Outline:
Assumption is that directly parsing the natural language to the parse tree is not appropriate as it may contain ambiguous and unstructured information. Therefore, they first represent the problem sentences into semantic representation of the Natural Language and move further. Modules are outlined below.

Approach contains three modules:
1) A meaning representation language called DOL was newly designed as the semantic representation of natural language text.

2) A semantic parser which transforms natural language sentences of a math problem into DOL representation.

3) A reasoning module to derive math expressions from DOL representation.

DOL: Meaning representation language
Every meaningful piece of NL text is represented in DOL as a semantic tree of various node types.
Node types of a DOL tree include constants, classes, and functions.

Constants: Constants in DOL refer to specific objects in the world.

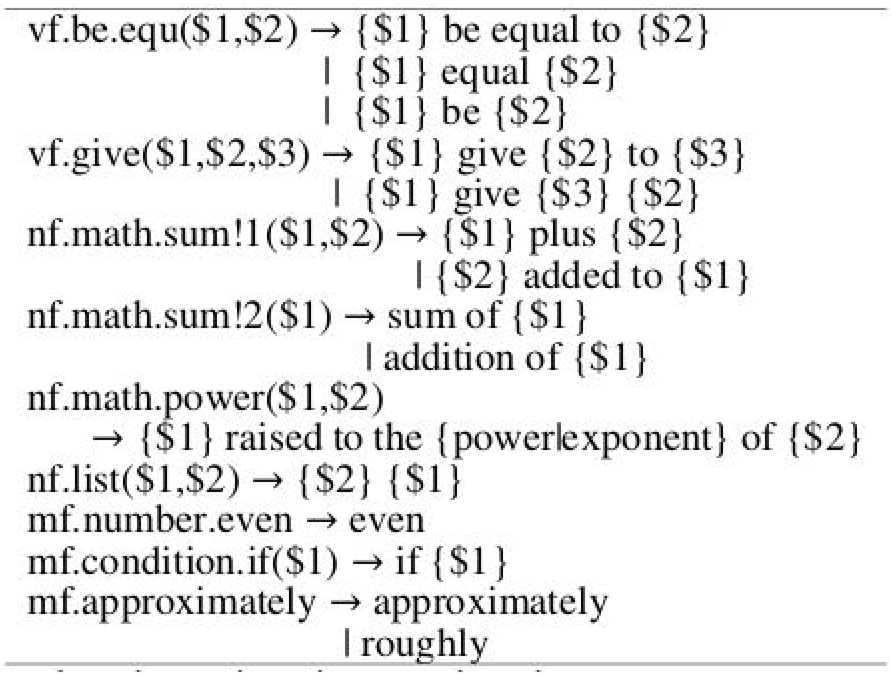
Classes: An entity class refers to a category of entities sharing common semantic properties. This can be a number class or a class defining name of the cities.

Functions: Functions are used in DOL as the major way to form larger language units from smaller ones. Eg : fn.math.sum. Which can use the relation between two sentences to generate the effect of the two sentences collectively.

(4) <u>Automatically Solving Number Word Problems by Semantic Parsing and Reasoning</u>

<u>CFG for connecting DOL and NL:</u>

The core part of a CFG is the set of grammar rules. Example English grammar rules for building syntactic parsers include “S → NP VP”, “NP →CD | DT NN | NP PP”, etc.] Below are some examples of Grammar rules,The left side of each rule is a DOL element(a function, class, or constant); and the right side is a sequence of words and arguments.



```
vf.be.equ($1,$2) → {$1} be equal to {$2}
                 | {$1} equal {$2}
                 | {$1} be {$2}
vf.give($1,$2,$3) → {$1} give {$2} to {$3}
                  | {$1} give {$3} {$2}
nf.math.sum!1($1,$2) → {$1} plus {$2}
                     | {$2} added to {$1}
nf.math.sum!2($1) → sum of {$1}
                  | addition of {$1}
nf.math.power($1,$2)
    → {$1} raised to the {power|exponent} of {$2}
nf.list($1,$2) → {$2} {$1}
mf.number.even → even
mf.condition.if($1) → if {$1}
mf.approximately → approximately
                 | roughly
```

<u>Reasoning:</u>

The reasoning module is responsible for deriving math expressions from DOL trees and calculating problem answers by solving equation systems. Pseudo code for reasoning after building the parse tree

```
Algorithm MathExpDerivation
   Input: DOL tree T
   Output: Math expression X(T)
   Global data structure: Expression list XL
1: For each child $C_i$ of T
2:     $X(C_i)$ = MathExpDerivation($C_i$)
3:     If X(Ci) is an s-expression
4:         Add $X(C_i)$ to XL
5: X(T) $\leftarrow$ Applying the semantic interpretation
of T
6: Return X(T)
```