Araştırma Makalesi / Research Article

An Object- Oriented Approach to Counter-Model Constructions in A Fragment of Natural Language

Selçuk TOPAL[*]

Bitlis Eren University, Faculty of Science and Arts, Department of Mathematics, Bitlis, Türkiye

Abstract
This article presents a technical construction of reasoning and counter-models for some sentences called fragments as in [9] in English. Speaking English and logical inferences are brought together in computer based approach to natural language. Not only the inferences in the language [7] are given but also counter-model constructions in case of no inference from input sentences. Approach of this construction considers usage of minimal number of set elements.

Keywords: Logic in Computer Science, Logic of Natural Languages, Applications of Logic

Doğal Dilin Bir Parçasındaki Karşıt Model Yapılarına Nesne Yönelimli Bir Yaklaşım

Özet
Bu makale, İngilizce dili içindeki [9] kaynağındaki gibi parçalar olarak adlandırılan bazı cümleler için türetimler ve karşıt modellerin bir teknik inşasını sunar. Günlük İngilizce ile mantıksal türetimleri doğal dile bilgisayar temelli yaklaşım içinde bir araya getirilmiştir. Sadece [7] dil içindeki türetimleri değil aynı zamanda girdi cümlelerinden bir türetim olmaması durumunda karşıt model inşaları da verilmiştir. Bu inşa yaklaşımı enaz sayıda küme elemanları kullanmayı göz önünde bulundurur.

Anahtar Kelimeler: Bilgisayar Biliminde Mantık, Doğal Dillerin Mantığı, Mantığın Uygulamaları

1. Introduction

Most of information and computer system focus on checking that queries correct or not. These queries are responded in the form of yes or no. We give algorithms and an illustrating implementation regarding the reason of a query is answered yes or no. This fragment has their origins in *Aristotle ‘syllogism*. *Aristotle* presented *syllogism* notion by helping approach of *categories* [2] and [12]. *Aristotelian syllogism* was begun to evaluate as an issue of formal logic by Lukasiewicz [6]. Corcoran gave a completeness theorem for Aristotelian syllogism as named “Completeness of an ancient logic” [5]. A modern completeness theorem was given by Moss [7]. Some complexity results of syllogistic sentences of English, completeness results of some syllogistic logics and algorithms and completeness results of some relational syllogistic logics were given in order of by [9], [7] and [10]. The fragment we consider in this paper is contained by [9] in view of complexity and by [7] in view of completeness. We integrate this fragment having efficient time complexity and logical completeness in to natural language. We give a general construction method in addition to we uploaded an instance of

[*]*Corresponding Author: s.topal@beu.edu.tr*

the construction script to sagemathcloud.com [14] to share it. Some algorithms shall be given in the syntax of Python [11] others in the pseudo-code for readability.

2. Preliminaries

Definition 2.1 [4] A context-free grammar (*CFG*) is a tuple $G = (M, \Sigma, R, S)$ where:

- M is a set of non-terminal symbols
- Σ is a set of terminal symbols
- R is a set of rules of the form $A \rightarrow B_1 \quad B_2 \ \ldots \ B_n$ such that $n \geq 0$, $A \in M$, $B_i \in (M \cup \Sigma)$
- $S \in M$ is a severalised start symbol

Example 2.2 A basic example in English for *CFG*:
M = {S, DT, Vt, NP, N}
Σ = {All, cats, are, animals}
S = S
R = : S → NP VP
VP → VBP NP
NP → DT N
DT → All
VBP → are
N → cats
N → animals

Definition 2.3 [7] Language of S starts by set P with p, q, r, ...variables (plural nouns) and a finite universe M. For every $p \in$ P, $[[p]] \subseteq M$ where [[ ]] is an *interpretation function* from P to subsets of M. A model M = (M, [[]], P) has the following truth properties:

$$\text{M} \vDash \textit{All p are q} :\Leftrightarrow [[p]] \subseteq [[q]]$$
$$\text{M} \vDash \textit{Some p are q} :\Leftrightarrow [[p]] \cap [[q]] \neq \emptyset$$
$$\text{M} \vDash \textit{No p are q} :\Leftrightarrow [[p]] \cap [[q]] = \emptyset$$

Table 1. Proof system for *S*

$$\frac{}{\mathit{All}\ x\ \mathit{are}\ x}\ (\text{axiom})$$

$$\frac{\mathit{No}\ x\ \mathit{are}\ x}{\mathit{All}\ x\ \mathit{are}\ y}\ (\text{no1}) \qquad \frac{\mathit{All}\ x\ \mathit{are}\ y \quad \mathit{All}\ y\ \mathit{are}\ z}{\mathit{All}\ x\ \mathit{are}\ z}$$

$$\frac{\mathit{No}\ x\ \mathit{are}\ y}{\mathit{No}\ y\ \mathit{are}\ x}\ (\text{Con}) \qquad \frac{\mathit{All}\ x\ \mathit{are}\ y \quad \mathit{No}\ y\ \mathit{are}\ z}{\mathit{No}\ x\ \mathit{are}\ z}\ (\text{AllNo})$$

$$\frac{\mathit{Some}\ x\ \mathit{are}\ y}{\mathit{Some}\ y\ \mathit{are}\ x}\ (\text{Com}) \qquad \frac{\mathit{All}\ x\ \mathit{are}\ y \quad \mathit{Some}\ x\ \mathit{are}\ z}{\mathit{Some}\ y\ \mathit{are}\ z}\ (\text{AllSome})$$

$$\frac{\mathit{Some}\ x\ \mathit{are}\ y \quad \mathit{No}\ x\ \mathit{are}\ y}{C}\ \mathcal{X}$$

The symbol *X* in Table 1 means that if Γ ⊢ *Some x are y* and Γ ⊢ *No x are y* then model is inconsistent. Every sentence in *S* can be derived from the model in the present case and *C* means any sentences in *S*.

Definition 2.4 [13] Reachability Problem in Directed Graph**:** Given a directed graph *G* = (*V*, *E*) and a vertex *v* in *G* which other vertices can be reached by a path starting from v.

Definition 2.5 [7] Let Γ be a set of sentences in *S*. A *proof tree* over Γ is a finite tree *T* whose nodes are labeled with sentences, and each node is either a leaf node labeled with an element of Γ, or else matches one of the rules in the proof system *S* in Table 1. Γ ⊢ *Φ* means that there is a proof tree *T* for over Γ whose root is labeled ***Φ***. We read this as Γ proves *Φ*, or Γ derives *Φ* or that *Φ* follows from Γ in our proof system *S*.

Example 2.6 A proof tree for given Γ = { Some p are q, All p are h, All h are m, All m are t } ⊢ Some p are t:

$$\dfrac{\dfrac{\dfrac{\text{Some p are q}}{\text{Some p are p}} \qquad \text{All p are h}}{\text{Some p are h}} \qquad \dfrac{\text{All h are m} \qquad \text{All m are t}}{\text{All h are t}}}{\text{Some p are t}}$$

Definition 2.7 [7] Let Γ be a finite set of *All* sentences. We say $p \rightarrow^{All} q$:⇔ Γ ⊢ *All p are q*. In other saying, there is a path from node *p* to node *q* if taking variables as nodes of graph obtained from Γ and $p\rightarrow^{All} q$ is a directed edge from *p* to *q* of the graph.

3. Derivation Algorithms in *S*

We take a set of input sentences Γ as a set of premises and Γ ⊢ *Φ* means that query sentence *Φ* is derivable from Γ. On the other hand, Γ ⊬ *Φ* means that *Φ* is not derivable from Γ.

Algorithm 1 An algorithm to check Γ ⊢ All p are q for given a finite set Γ ⊆ *S*.

```
1: If there is a path from node $p$ to $q$ Then Print the path
2: If No p are p in Γ Then Print All p are q from the rule (no1)
3: If Γ is inconsistent Then Print Γ⊢ All p are q from the rule $X$
4: Else Counter-Model
```

Algorithm 2 An algorithm to check Γ ⊢ Some p are q for given a finite set Γ ⊆ *S*.

```
1: If {All n are p ∈ Γ or Γ ⊢ All n are p} and {Γ ⊢ All m are q or All m are q ∈ Γ} and
   {Some m are n ∈ Γ or Some n are m ∈ Γ} Then Print Some p are q from the rule (All, Some)
   [11].
2: If Γ is inconsistent Then Print Γ⊢ Some p are q from the rule $X$
3: Else Counter-Model
```

Algorithm 3 An algorithm to check Γ ⊢ No p are q for given a finite set Γ ⊆ *S*.

```
1: If {All p are n ∈ Γ or Γ ⊢ All p are n} and {Γ ⊢ All q are m or All q are m ∈ Γ} and
   {No m are n ∈ Γ or No n are m ∈ Γ} Then Print No p are q from the rule (All, No) [11]
2: If Γ is inconsistent Then Print Γ ⊢ No p are q from the rule $X$
3: Else Counter-Model
```

Algorithm 4 An algorithm to check Γ ⊢ No p are q and Γ ⊢ Some p are q for given a finite set Γ ⊆ *S*.

```
1: If Algorithm 2 satisfies $\Gamma \vdash$ Some p are q and Algorithm 3 satisfies $\Gamma \vdash$ No p are q except
   inconsistencies Then Print $\Gamma$ is inconsistent.
```

4. Counter-Model Constructions in *S*

We use sets in order to construct counter-models since *S* logic has set-theoretic model. $[[p]]$ and $[[q]]$ have to have at least one common element since Some p are q means $[[p]] \cap [[q]] \neq \emptyset$.We prefer to assign $\{p,q\}$ to both $[[p]]$ and $[[q]]$ in order that they have a common element (see Algorithm 5).

Algorithm 5 An algorithm for assigning set values to variables of input sentences in *S*.

```
1: If the input is All p are q then $[[p]] \leftarrow \emptyset$ and $[[q]] \leftarrow \emptyset$
2: If the input is Some p are q then $[[p]] \leftarrow \{p1, p2, \{p,q\}\}$ and $[[q]] \leftarrow \{q1, q2, \{p,q\}\}$
3: If the input is No p are q then, if $p = q$ then $[[p]] = [[q]] \leftarrow \emptyset$ Else $[[p]] \leftarrow \{p1\}$ and
   $[[q]] \leftarrow \{q1\}$.
```

$[[p]]$ and $[[q]]$ has not any common elements since No p are q means $[[p]] \cap [[q]] = \emptyset$. If not $\Gamma \vdash$ No p are p or not $\Gamma \vdash$ then we do not know $[[p]] = \emptyset$ or $[[q]] = \emptyset$ accurately. Thus we make assignment $[[p]] = \{p1\}$ and $[[p]] = \{p2\}$ (see Algorithm 6).

Algorithm 6 An algorithm for constructing steps of counter-models for queries that are not derived from input set in *S*.

```
1: If $\Gamma \nvdash$ All p are q Then $[[q]] \leftarrow [[q]] \cup \{q1\}$ and $[[p]] \leftarrow [[p]] \cup \emptyset$
2: If $\Gamma \nvdash$ No p are q Then $[[q]] \leftarrow [[q]] \cup \{q1, q2, \{p,q\}\}$ and $[[p]] \leftarrow [[p]] \cup \{p1,p2,\{p,q\}\}$
3: If $\Gamma \nvdash$ Some p are q Then $[[q]] \leftarrow [[q]] \cup \emptyset$ and $[[p]] \leftarrow [[p]] \cup \emptyset$
4:
```

Algorithm 7 An algorithm for updating process of model to construct counter-model in *S*.

```
1: $[[q]] \leftarrow [[q]] \cup [[p]]$ for all variable $p \in P_{All} \cap P_{Some}$ and for all variable $q \in P$ where $\Gamma \vdash$ All p
   are q If $\Gamma \nvdash$ No p are q Then $[[q]] \leftarrow [[q]] \cup \{q1, q2, \{p,q\}\}$ and $[[p]] \leftarrow [[p]] \cup$
   $\{p1,p2,\{p,q\}\}$
2: $[[q]] \leftarrow [[q]] \cup [[p]]$ for all $p \in P_{All} \cap P_{No}$ and for all variable $q \in P$ where $\Gamma \vdash$ All p are q
3: $[[q]] \leftarrow [[q]] \cup [[p]]$ for all variable $p \in P_{Some}$ but $p \notin P_{All}$ and for all variable $q \in P$ where
   $\Gamma \vdash$ All p are q
4: $[[q]] \leftarrow \emptyset$ for all variables $p \in P$ but $p \notin P_{No}$ and for all variable $q \in P$ where $\Gamma \vdash$ No p are p
   and $\Gamma \vdash$ All q are p
```

5. Integrating Algorithms of *S* with Natural Language

In this section, we consider how to be detected whether a sentence is or not in the grammar of language of *S*. We use certain properties of NLTK module of Python Program [3] to do this. We prefer to utilize the tools of POS-tagger function and WordNet package in NLTK for the purpose of checking sentences whether to be or not in natural spoken English and the grammar of language of *S*.

The function POS-tagger determines syntactic symbols of words of a sentence. Therefore the function provides to specify the grammar of intended sentences which are as in Figure 1 by using tree and tagging functions of part of speech tagging as in Table 2. WordNet is a lexical database for English. We use WordNet in order to test whether words of input sentences in language of *S* is or not in English. WordNet package which is provided by NLTK serves online query for English words.

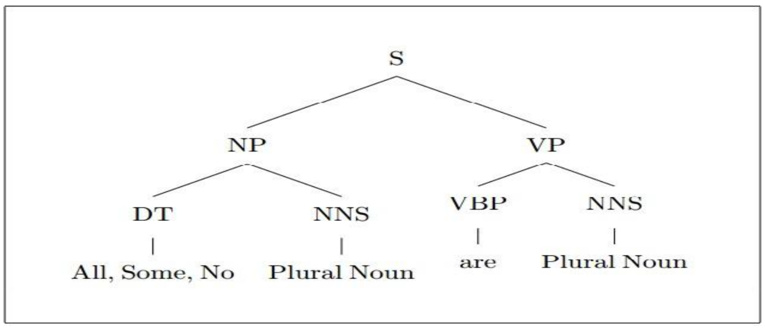


Figure 1. A tree for the grammar

In Algorithm 8, input sentences are strings and **text ← nltk.word.tokenize (*Φ*)** means that the program turns the string *Φ* into a list text such that a list is a data structure in Python (see Figure 2). **tagging ← nltk.postag (text)** means that the program turns the list text into the nested tagging such that nested list is a list data structure in Python (see Figure 3). If a input string belongs to WordNet, it belongs to spoken English, in other saying, natural English. Equivalent of this fact is that the function **wn.synset (*α*)** returns a list data structure if it finds a data for the string *α* in WordNet and bool value of a list is **True** in Python (see figure 4).

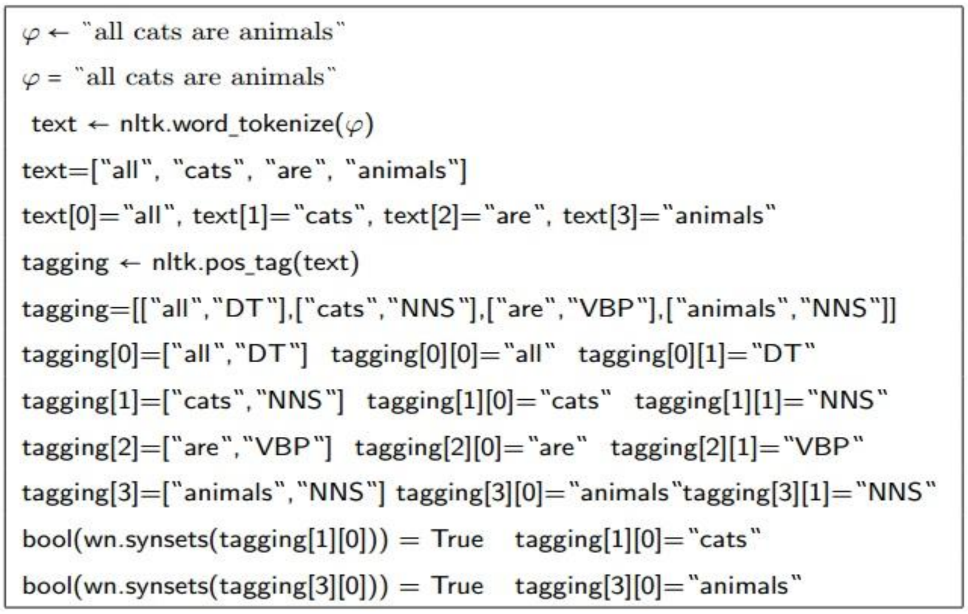
Table 2. Part of speech for the grammar

Syntactic Symbol	Meaning
NNS	Plural Noun
DT	Determiner
VBP	Verb, non-3rd person singular present

Algorithm 8 An algorithm for detections of sentences by using NLTK module.

```
1:  While True do
2:  Read the sentence Φ
3:  Φ ← ``sentence ``                     ▷ sentence is a string
4:  If Φ==``no``  Then Break              ▷ == means equality testing in Python
5:  text ← nltk.word tokenize(Φ)          ▷ parsing the sentence word by Word
6:  tagging ← nltk.pos tag(text)          ▷ labeling each words by POS-tagger
7:  If tagging[0][1] != ``DT`` Or bool(wn.synsets(tagging[0][0]))!=True Or tagging[0][0] in
    (``all``, ``some``, ``no``) Then Break     ▷ != means non-equality testing in Python
8:  If tagging[1][1] !=``NNS `` Or bool(wn.synsets(tagging[1][0])) !=True Then Break
9:  If tagging[2][1]!=``VBP`` Or bool(wn.synsets(tagging[2][0])) !=True Then Break
10: If tagging[3][1] !=``NNS`` Or bool(wn.synsets(tagging[3][0])) !=True Then Break
11: End While
```

Figure 2 illustrates how Algorithm 8 works for an input sentence **all cats are animals**. The sentence **all cats are animals** is segmented word by word. Words **all, cats, are** and **animals** are tagged with non-terminal symbols. The words **cats** and **animals** are checked with WordNet whether they are in English or not.



```
φ ← "all cats are animals"
φ = "all cats are animals"
 text ← nltk.word_tokenize(φ)
text=["all", "cats", "are", "animals"]
text[0]="all", text[1]="cats", text[2]="are", text[3]="animals"
tagging ← nltk.pos_tag(text)
tagging=[["all","DT"],["cats","NNS"],["are","VBP"],["animals","NNS"]]
tagging[0]=["all","DT"]  tagging[0][0]="all"  tagging[0][1]="DT"
tagging[1]=["cats","NNS"]  tagging[1][0]="cats"  tagging[1][1]="NNS"
tagging[2]=["are","VBP"]  tagging[2][0]="are"  tagging[2][1]="VBP"
tagging[3]=["animals","NNS"] tagging[3][0]="animals"tagging[3][1]="NNS"
bool(wn.synsets(tagging[1][0])) = True   tagging[1][0]="cats"
bool(wn.synsets(tagging[3][0])) = True   tagging[3][0]="animals"
```

Figure 2. An Illustration of Algorithm 8 for ``all cats are animals``

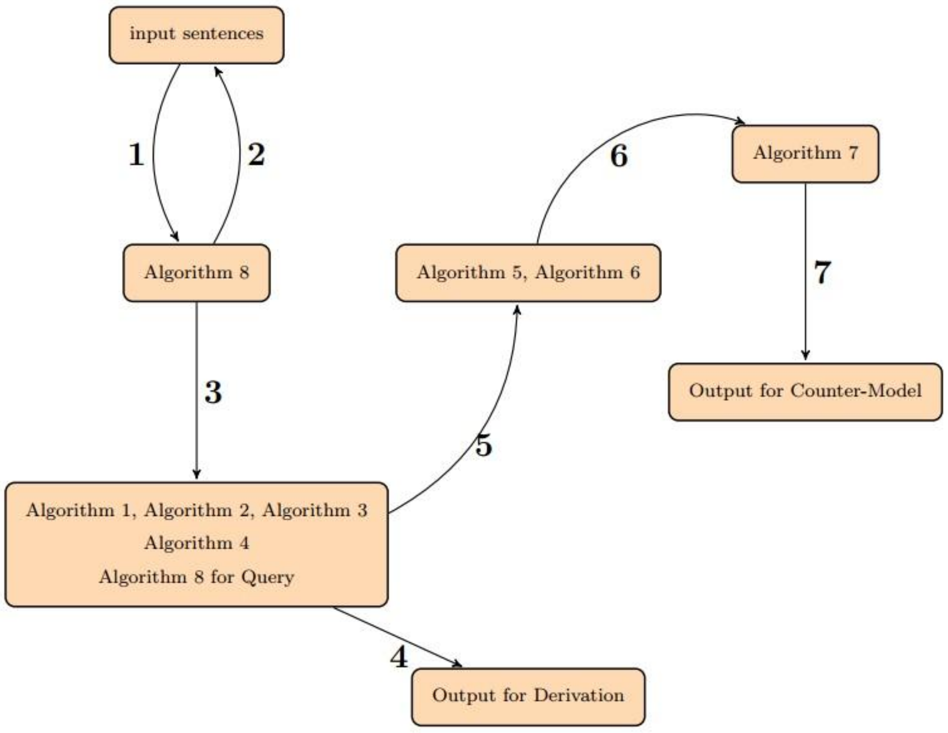


Figure 3. Flow-chart with algorithms for the system

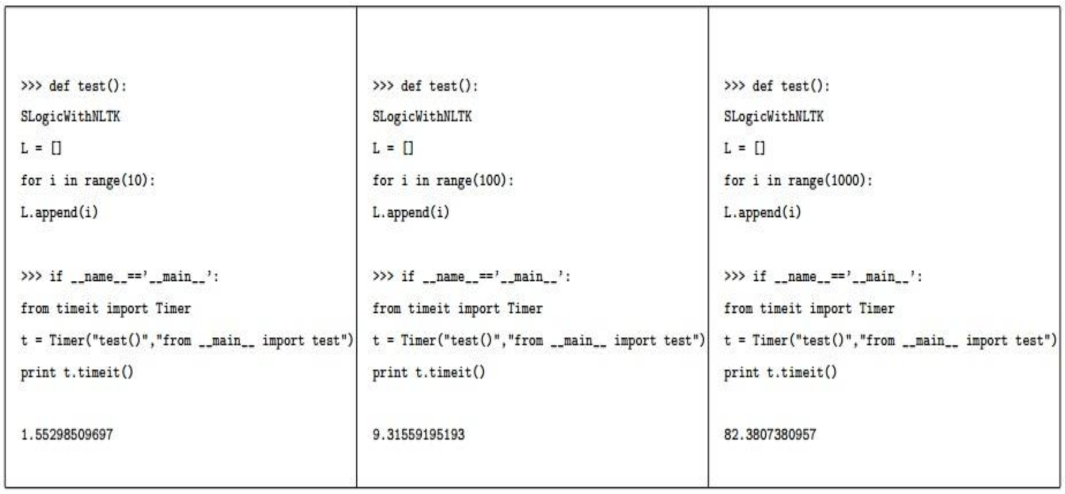
Figure 3 illustrates how entire system works. Arrows in the figure provides transition among used algorithms and decisions. Algorithm 8 is used two times in order to check the grammar of input sentences and query sentence. Tasks of the arrows are the following in detail:

Arrow 1: Sending input sentences to NLTK module.
Arrow 2: Asking back a sentence due to unaccepted sentences format.
Arrow 3: Sending accepted sentences for asking a query and checking for a derivation.
Arrow 4: If the derivation occur then sending it to output for visualization of proof.
Arrow 5: If the derivation does not occur then sending variables to the constructing process to assign set values to them.
Arrow 6: Sending the input and the query to counter-model construction for updating set values and entire model.
Arrow 7: Sending updated model for visualization.

5.1. Technical Details

In this section, we consider a graph of input number-time (seconds) comparison of the implementation that is implemented for this research. The implementation can be found as a project S Logic with Counter-Model and NLTK on Sagemath Cloud [14] if request an access to the project. We here test the script how much time (in seconds) to run for 10, 100 and 1000 input. We take the script as a Python function to test on 8GB RAM, 64-bit operating system and 2.40 Ghz CPU.

Table 3. Running time of the script for 10, 100 and 1000 input number



```
>>> def test():
SLogicWithNLTK
L = []
for i in range(10):
L.append(i)

>>> if __name__==’__main__’:
from timeit import Timer
t = Timer("test()","from __main__ import test")
print t.timeit()

1.55298509697
```

```
>>> def test():
SLogicWithNLTK
L = []
for i in range(100):
L.append(i)

>>> if __name__==’__main__’:
from timeit import Timer
t = Timer("test()","from __main__ import test")
print t.timeit()

9.31559195193
```

```
>>> def test():
SLogicWithNLTK
L = []
for i in range(1000):
L.append(i)

>>> if __name__==’__main__’:
from timeit import Timer
t = Timer("test()","from __main__ import test")
print t.timeit()

82.3807380957
```

Ordered pairs of input number - time values are (10, 1.55298509697), (100, 9.31559195193), (1000, 82.3807380957) as can be seen in Table 3. Function from these ordered pairs is approximated the function $f(x) \approx -0.001x^2+0.086x+0.682$ obtained by using Lagrange interpolation for three points.

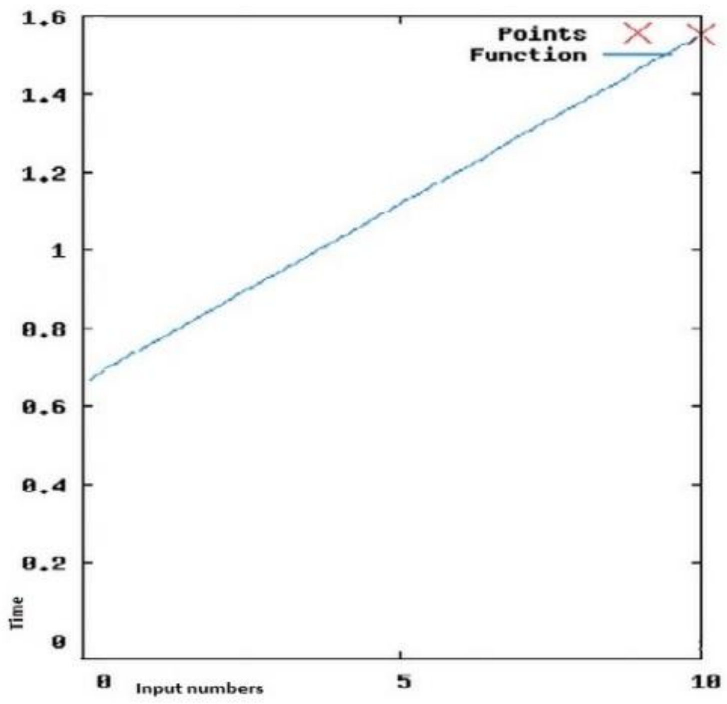


Figure 4. Input number-time match graph for SLogicWithNLTK implementation

Graph of the function in Figure 4 shows that the function behaves nearly $f(x) \approx 10x + 12$. The script has a very efficient run time under reasonable input number.

Remark 1 We could not find any comparable work to compare to or with our work in literature since this research is in a very new research area and multidiscipline.

6. Future Work

We will implement a logic [8] which has richer language, grammar and expressive power than *S* logic has.

References

1. Andrade EJ, Becerra E, 2007. Corcoran's Aristotelian syllogistic as a subsystem of first-order logic. Revista Colombiana de Matem`aticas, 41(1): 67-80.
2. Aristotle Categories, 1984. The complete works of Aristotle, Princeton University Press, vol. 1, 1256 p.
3. Bird S, 2006. NLTK: the natural language toolkit. In Proceedings of the COLING/ACL on Interactive presentation sessions, Association for Computational Linguistics, pp. 69-72.
4. Hopcroft JE, Motwani R, Ullman JD, 2001. Introduction to Automata Theory, Languages and Computation. ACM SIGACT News, 32 (1): 60-65.
5. Corcoran J, 1972. Completeness of an ancient logic. Journal of Symbolic Logic, 37: 696-702.
6. Lukasiewicz J, 1951. Aristotle's syllogistic from the standpoint of modern formal logic. Clarendon Press, Oxford.

7. Moss LS, 2008. Completeness theorems for syllogistic fragments, eds. Logics for linguistic structures, Walter de Gruyter, 201: 143-175.
8. Moss LS, 2011. Syllogistic logic with complements, Games, Norms and Reasons. Springer Netherlands, 179-197.
9. Pratt-Hartmann I, 2004. Fragments of language, Journal of Logic, Language and Information, 13.2: 207-223.
10. Pratt-Hartmann I, Moss LS, 2009. Logics for the Relational Syllogistic, The Review of Symbolic Logic, 2.04: 647-683.
11. Python Software Foundation, Python Language Reference, version 2.7. Available at http://www.python.org.
12. Smith R, 1989. Categories, Aristotle's Prior Analytics, Hackett Publishing Company, 320p.
13. Cormen TH, Leiserson CE, Rivest RL, 1989. Introduction to Algorithms. The MIT Press and McGraw-Hill Book Company, 597p.
14. Sage Mathematics Software, 2015. The Sage Development Team, http://www.sagemath.org

Geliş Tarihi: 20/04/2015
Kabul Tarihi: 31/07/2015