

# Metacognitive Question Answering from Euclid's Elements Text

John Kontos and Joseph Armaos

**Abstract--** The present paper presents a system for meta-cognitive question answering from texts as applied to the Euclid's "Elements" text. The system can answer "why" questions from geometric proof texts by representing and processing its own state of automatic proof analysis and generating explanations and supporting diagrams. The system is implemented in Prolog and uses a question grammar combined with a text grammar. These two grammars use rules, lexicons, ontologies and the state of the system to answer the input questions. The internal recording of the state of the system mainly in terms of the strategy being followed is the main manifestation of its metacognitive architecture. The semantics of the question grammar are expressed in terms of combinations of predicates that analyze chains of text chunks recognized by the text grammar. Inference of implicit facts is performed directly from the text. The system facilitates adaptation to the language of different users.

**Index Terms** – Question Answering, Meta-cognition, Geometry.

## 1 INTRODUCTION

THE question answering system presented in the present paper is based on an approach that aims at the creation of systems which accept natural language questions and generate answers with information extracted from a text and a record of its own state either directly or after applying deductive inference and meta-information concerning the state of the system. The internal record of the state of the system contains information concerning the strategy followed and the progress of the deductive inference that is used for generating automatically explanations of the answers given to the input questions. The strategies available to the system are explicitly listed and consist of combinations of tactics such as:

- Focusing on and transforming different parts of the question.
- Using synonyms of entity names.
- Using prerequisite knowledge such as common notions.

The deductive inference from texts by computer is traditionally performed in two stages. In the first stage the text is translated by computer or by hand into some formal representation. In the second stage reasoning is performed with this formal representation. Contrary to that we avoid the translation step and deduction is performed directly from the texts following the ARISTA method of text analysis. This method was first proposed in [3] and further elaborated in [4,5,6 and 7]. The main advantage of this method as applied to scientific text is that there is no need for retranslation whenever a change is made in the ontology of the domain or the meaning of its technical terms. The present system answers questions deductively from the English translation of the texts of the proofs of the Elements of Euclid [2].

The progress of the deductive inference in our system can be monitored by following and recording the use of the sentences of the proof text and the common notions text that supports the generation of the explanations provided as deductive answers to the "why" questions of the user. This monitoring together with the self-awareness of the strategy being followed constitutes the main elements of the metacognitive nature of our question answering system.

## 2 THE STRATEGIES FOR PROCESSING THE QUESTIONS

*Department of Philosophy & History of Science  
Artificial Intelligence Group  
NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS  
HELLAS E-mail: [ikontos2003@yahoo.com](mailto:ikontos2003@yahoo.com), [iarmaos@otenet.gr](mailto:iarmaos@otenet.gr) )*

The questions processed by our system concern the English translation [2] of the proofs of the elements of Euclid based on different strategies.

E.g. the first proof concerns the constructive proposition "On a given finite straight line to construct an equilateral triangle".

If we submit the question "why ca is equal to bc" we expect to get an answer giving an explanation using the text of the proof for the correctness of the proposition stating the fact "ca is equal to bc". Note that ca and bc are straight line segments of the construction. Some of the questions concern facts that are proved by simple reference to another fact. Some other questions require use of prerequisite knowledge for answering them. Following some of the principles of metacognition found in experiments of educational psychology we use an explicit list of strategies for processing the questions before answering them and for exploiting prerequisite knowledge. The prerequisite knowledge we use involves various kinds such as Euclid's common notions, postulates and definitions. Other sources of knowledge are a list of synonyms of the entities involved. A strategy is defined by the predicate strategy (Synonym, LSide, RSide, Inversion, Relstrategy, Pk, strategy1). The roles and the possible values of the variables for each argument of the predicate "strategy" are:

| Variable Name | Role                   | Possible Values               |
|---------------|------------------------|-------------------------------|
| Synonym       | flag for synonymes     | nosynonymes, mesynonymes      |
| LSide         | flag for left side     | left, noleft                  |
| RSide         | flag for right side    | right, noright                |
| Inversion     | flag for inversion     | noinversion, meinversion      |
| Relstrategy   | relation specification | equality, meronymy, causality |
| Pk            | prerequisite knowledge | Pk, nopk                      |
|               | flag                   |                               |
| Strategy      | strategy name          | strategy1, strategy2, etc     |

These strategies can be applied to the question answering from text with content different than Geometry. The operation of the system will be explained by showing how the question "why ac is equal to ab" is processed by the system.

The answering of this question is found by following strategy1. This strategy uses the expression of the question as it stands without any transformation. It should be noted that no prerequisite knowledge is used when strategy1 is followed. The answering of other questions may follow other strategies that apply various transformations of the questions such as inversion of its terms and replacement of an entity by each synonym. More specifically the inversion of "ac is equal to ab" is "ab is equal to ac" and the synonym of "ac" is "ca". The full list of the 16 strategies used in the present form of our system is:

```
strategy(nosynonymes, left, right, noinversion, equality, nopk, strategy1).
strategy(nosynonymes, left, right, meinversion, equality, nopk, strategy2).
strategy(mesynonymes, left, noright, noinversion, equality, nopk, strategy3).
strategy(mesynonymes, left, noright, meinversion, equality, nopk, strategy4).
strategy(mesynonymes, noleft, right, noinversion, equality, nopk, strategy5).
strategy(mesynonymes, noleft, right, meinversion, equality, nopk, strategy6).
strategy(mesynonymes, left, right, noinversion, equality, nopk, strategy7).
strategy(mesynonymes, left, right, meinversion, equality, nopk, strategy8).
strategy(nosynonymes, left, right, noinversion, equality, pk, strategy9).
strategy(nosynonymes, left, right, meinversion, equality, pk, strategy10).
strategy(mesynonymes, left, noright, noinversion, equality, pk, strategy11).
strategy(mesynonymes, left, noright, meinversion, equality, pk, strategy12).
strategy(mesynonymes, noleft, right, noinversion, equality, pk, strategy13).
strategy(mesynonymes, noleft, right, meinversion, equality, pk, strategy14).
strategy(mesynonymes, left, right, noinversion, equality, pk, strategy15).
strategy(mesynonymes, left, right, meinversion, equality, pk, strategy16).
```

The name of the current strategy is stored in an internal data base so that the system is aware of the strategy being followed each time. The chains of the proof steps as mentioned in the proof text are also stored so that may be used at explanation time. The first step is to call the rule

question(Question) which follows :

```
question(Question):-retractall(cn1(_)),
                    extract_question(Question),
                    strategy(Synonym,LSide,RSide,Inversion,Relstrategy,Pk,strategy1),
                    retractall(oldstrategy(_)),
                    assert(oldstrategy(strategy1)),
                    nl,write("TRYING STRATEGY : strategy1"),
                    write("with",Synonym,"_",LSide,RSide,Inversion,Relstrategy,Pk),
                    process_answer(strategy1).
```

```
question(Question):-extract_question(Question),
                    oldstrategy(OldS),
                    next(OldS,Strategy),
                    questionloop(Strategy).
```

```
question(Question):-nl,write("impossible to answer",Question),nl.
```

The first alternative applies strategy1, resets the common notion flag cn1 and then calls the rule `extract_question(Question)` that is presented in the next section. If strategy1 fails the second alternative applies strategy2 and so on using the rules “next” and “questionloop”.

### 3. EXTRACTION OF THE COMPONENTS OF THE QUESTIONS

```
extract_question(Question):-f(Question,why,Rest1),
                             f(Rest1,LEntity,Rest2),
                             f(Rest2,is,Rest3),
                             f(Rest3,Relation,Rest4),
                             f(Rest4,Prep,Rest5),
                             f(Rest5,REntity,""),
                             relation(Relation),preposition(Prep),
                             retractall(leftentityofquestion(_)),
                             assert(leftentityofquestion(LEntity)),
                             retractall(relationofquestion(_)),
                             assert(relationofquestion(Relation)),
                             retractall(rightentityofquestion(_)),
                             assert(rightentityofquestion(REntity)).
```

The input question is syntactically analysed with the above rule using the Prolog function “fronttoken” which here is symbolised by “f” for brevity. The function “f” has as first argument a string e.g. the question string “why ac is equal to ab” and returns the first word which is the word “why” as second argument and as third argument the rest of the string namely “ac is equal to ab”.

This analysis extracts the components of the question (`LEntity`, `REntity`) in this case “ac” and “ab” as well as the word denoting the relation between them namely “equal”. These components are stored in the internal data base. After the completion of the extraction rule the generation of the answer to the question is accomplished using the rule `process_answer(Strategy)` following the current strategy which for the present example is strategy1. If a rule is not satisfied then a new strategy is tried by executing the second alternative of the rule:

```
question(Question):-extract_question(Question),
                    oldstrategy(OldS),
                    next(OldS,Strategy),
                    questionloop(Strategy).
```

The predicate `next(OldS,Strategy)`, finds the next strategy to try after the strategy OldS that failed. The sequencing of the strategies is defined by the predicate “next” and constitutes what we call a “policy”. The recursive rule `questionloop(Strategy)` applies the new strategy and if that strategy also fails it tries the next one and so on until all strategies are applied.

This rule has as follows:

```
questionloop(Strategy):-strategy(Synonym,LSide,RSide,
                                Inversion,Relstrategy,Pk,Strategy),
                                retractall(oldstrategy(_)),
                                assert(oldstrategy(Strategy)),
                                process_answer(Strategy).
questionloop(_):-oldstrategy(OLDS), next(OLDS,NewStrategy),
                questionloop(NewStrategy).
```

#### 4. THE TRANSFORMATION OF THE QUESTIONS

The rule process\_answer mentioned above synthesizes transforms of the question following the current strategy by calling the alternative of the rule "compose" which is given below that corresponds to the current strategy. Note that each alternative may correspond to two strategies that differ only in the value of the Pk variable explained in the table shown above.

```
compose_question(CQ,S,I):-
    S="nosynonymes", I="noinversion",
    leftentityofquestion(LEntity),
    relationofquestion(Relation),
    rightentityofquestion(REntity),
    c(" ",LEntity,BLE),c(BLE," ",T1),
c("is","_",Isb),c(Isb,Relation,IsbR),c(IsbR,"_",IsbRb),c(IsbRb,"to",T2),
c(" ",REntity,BRE),c(BRE," ",T3), c(T1,T2,T12),c(T12,T3,CQ),
    retractall(leftentityofquestiontransferred(_)),
    assert(leftentityofquestiontransferred(LEntity)),
    retractall(rightentityofquestiontransferred(_)),
    assert(rightentityofquestiontransferred(REntity)).
```

```
compose_question(CQ,S,I):-
    S="nosynonymes", I="meinversion",
    leftentityofquestion(LEntity),
    relationofquestion(Relation),
    rightentityofquestion(REntity),
    c(" ",LEntity,BLE),c(BLE," ",T1),
c("is","_",Isb),c(Isb,Relation,IsbR),c(IsbR,"_",IsbRb),c(IsbRb,"to",T2),
c(" ",REntity,BRE),c(BRE," ",T3), c(T3,T2,T32),c(T32,T1,CQ),
    retractall(leftentityofquestiontransferred(_)),
    assert(leftentityofquestiontransferred(REntity)),
    retractall(rightentityofquestiontransferred(_)),
    assert(rightentityofquestiontransferred(LEntity)).
```

```
compose_question(CQ,S,L,I):-
    S="mesynonymes",L="left",I="noinversion",
    leftentityofquestion(LEntity),
    relationofquestion(Relation),
    rightentityofquestion(REntity),
    synwnyma(LEntity,SLEntity),
    c(" ",SLEntity,BLE),c(BLE," ",T1),
c("is","_",Isb),c(Isb,Relation,IsbR),c(IsbR,"_",IsbRb),c(IsbRb,"to",T2),
c(" ",REntity,BRE),c(BRE," ",T3), c(T1,T2,T12),c(T12,T3,CQ),
    retractall(leftentityofquestiontransferred(_)),
    assert(leftentityofquestiontransferred(SLEntity)),
    retractall(rightentityofquestiontransferred(_)),
    assert(rightentityofquestiontransferred(REntity)).
```

```

compose_question(CQ,S,L,I):-
    S="mesynonymes",L="left",I="meinversion",
    leftentityofquestion(LEntity),
    relationofquestion(Relation),
    rightentityofquestion(REntity),
    synwnyma(LEntity,SLEntity),
    c(" ",SLEntity,BLE),c(BLE," ",T1),
c("is","_",Isb),c(Isb,Relation,IsbR),c(IsbR,"_",IsbRb),c(IsbRb,"to",T2),
c(" ",REntity,BRE),c(BRE," ",T3), c(T3,T2,T32),c(T32,T1,CQ),
    retractall(leftentityofquestiontransferred(_)),
    assert(leftentityofquestiontransferred(REntity)),
    retractall(rightentityofquestiontransferred(_)),
    assert(rightentityofquestiontransferred(SLEntity)).

```

```

compose_question(CQ,S,R,I):-
    S="mesynonymes",R="right",I="noinversion",
    leftentityofquestion(LEntity),
    relationofquestion(Relation),
    rightentityofquestion(REntity),
    synwnyma(REntity,SREntity),
    c(" ",LEntity,BLE),c(BLE," ",T1),
c("is","_",Isb),c(Isb,Relation,IsbR),c(IsbR,"_",IsbRb),c(IsbRb,"to",T2),
c(" ",SREntity,BRE),c(BRE," ",T3), c(T1,T2,T12),c(T12,T3,CQ),
    retractall(leftentityofquestiontransferred(_)),
    assert(leftentityofquestiontransferred(LEntity)),
    retractall(rightentityofquestiontransferred(_)),
    assert(rightentityofquestiontransferred(SREntity)).

```

```

compose_question(CQ,S,R,I):-
    S="mesynonymes",R="right",I="meinversion",
    leftentityofquestion(LEntity),
    relationofquestion(Relation),
    rightentityofquestion(REntity),
    synwnyma(REntity,SREntity),
    c(" ",LEntity,BLE),c(BLE," ",T1),
c("is","_",Isb),c(Isb,Relation,IsbR),c(IsbR,"_",IsbRb),c(IsbRb,"to",T2),
c(" ",SREntity,BRE),c(BRE," ",T3), c(T3,T2,T32),c(T32,T1,CQ),
    retractall(leftentityofquestiontransferred(_)),
    assert(leftentityofquestiontransferred(SREntity)),
    retractall(rightentityofquestiontransferred(_)),
    assert(rightentityofquestiontransferred(LEntity)).

```

```

compose_question(CQ,S,L,R,I):-
    S="mesynonymes",L="left",R="right",I="noinversion",
    leftentityofquestion(LEntity),
    relationofquestion(Relation),
    rightentityofquestion(REntity),
    synwnyma(LEntity,SLEntity),
    synwnyma(REntity,SREntity),
    c(" ",SLEntity,BLE),c(BLE," ",T1),
c("is","_",Isb),c(Isb,Relation,IsbR),c(IsbR,"_",IsbRb),c(IsbRb,"to",T2),
c(" ",SREntity,BRE),c(BRE," ",T3), c(T1,T2,T12),c(T12,T3,CQ),
    retractall(leftentityofquestiontransferred(_)),
    assert(leftentityofquestiontransferred(SLEntity)),
    retractall(rightentityofquestiontransferred(_)),
    assert(rightentityofquestiontransferred(SREntity)).

```

```

compose_question(CQ,S,L,R,I):-
    S="mesynonymes",L="left",R="right",I="meinversion",
    leftentityofquestion(LEntity),
    relationofquestion(Relation),
    rightentityofquestion(REntity),
    synwnyma(LEntity,SLEntity),
    synwnyma(LEntity,SLEntity),
    synwnyma(REntity,SREntity),
    c(" ",SLEntity,BLE),c(BLE," ",T1),
c("is","_",Isb),c(Isb,Relation,IsbR),c(IsbR,"_",IsbRb),c(IsbRb,"to",T2),
c(" ",SREntity,BRE),c(BRE," ",T3), c(T3,T2,T32),c(T32,T1,CQ),
    retractall(leftentityofquestiontransferred(_)),
    assert(leftentityofquestiontransferred(SREntity)),
    retractall(rightentityofquestiontransferred(_)),
    assert(rightentityofquestiontransferred(SLEntity)).

```

The various parts of the question are combined by the rule presented above using the Prolog function concat which here is symbolised by c for brevity. The function c has as first and second argument two strings that their concatenation is returned as the third argument e.g. if we write c(A,B,C), then C is the concatenation of the two strings A and B.

## 5. THE GENERATION OF THE ANSWER FROM THE PROOF TEXT

After the execution of the one of the above alternatives the transformed question is compared with the sentences of the text of the proof of the proposition. When a match is found an answer is generated giving an explanation that justifies the answer. This procedure is accomplished using the rule "same\_relation" given below:

```

same_relation:-leftentityofquestiontransferred(LEntity),
    rightentityofquestiontransferred(REntity),
    number_leftpart_verb(LEntity,N),
    find_rightpart_verbless(N,Rightpartnextverb),
    f(Rightpartnextverb,equal,T1),
    f(T1,to,T2),
    f(T2,Rightentity," "),
    Rightentity=REntity.

```

The transformed parts of the question are retrieved by the rule presented above from the internal data base. By calling the rule "number-leftpart-verb" rule given below:

```

number_leftpart_verb(LEntity,N):-openread(verbless,"verbless.pro"),
    readdevice(verbless),
    c(" ",LEntity,BL),
    match_leftpart_verbless(BL),
    numberleftpart(N),
    closefile(verbless),
    readdevice(keyboard).

```

The program finds the position of the left part of a proof step that matches the left part of the transformed question using the rule match\_leftpart\_verbless(BL) and continues by matching the relation and the right part of the question with some step of the proof. This matching is accomplished by the following rule:

```

match_leftpart_verbless(BL):-readint(N),
    retractall(numberleftpart(_)),
    assert(numberleftpart(N)),
    N<>0,readln(Leftentity),
    retractall(leftpartverbless(_)),
    assert(leftpartverbless(Leftentity)),
    readln(_),readln(_),
    leftpartverbless(Leftentity),
    BL<>Leftentity,
    match_leftpart_verbless(BL).

```

```

match_leftpart_verbless(_):-numberleftpart(N),
                             N<>0,
                             closefile(verbless),
                             readdevice(keyboard).

match_leftpart_verbless(_):-numberleftpart(N),N=0,
                             closefile(verbless),
                             readdevice(keyboard),fail.

```

Following the satisfaction of the rule "same\_relation" the system returns to the rule process\_answer in order to continue the generation of the answer. The satisfaction of the same\_relation rule means that a sentence is found in the text of the proof that matches the question. When this happens then the sentence previous to the matched one is included in the answer provided that it is not the statement of some common notion. In the latter case the common is applied in order to generate a more complex answer.

## 6 AN ILLUSTRATIVE EXAMPLE OF THE SYSTEM PERFORMANCE

An illustrative example from the first Proposition of the Elements is given below together with the explanatory answers generated by the system for a few illustrative questions. The first Proposition may be stated in English: "On a given finite straight line to construct an equilateral triangle". The equilateral triangle abc is created by the point of intersection of two equal circles and their centers a and b where these centers lie on each others circumference and the points a and b are the two endpoints of the given finite straight line. Three illustrative questions are answered by the system by analyzing automatically the corresponding Euclidean proof as follows:

|          |  |
|----------|--|
| Question | 1: "why is side ac equal to side ab?"            |
| Answer   | 1: "because they are radii of the same circle 1" |
| Question | 2: "why is side bc equal to side ba?"            |
| Answer   | 2: "because they are radii of the same circle 2" |
| Question | 3: "why is side ca equal to side cb?"            |
| Answer   | 3: "because each of ca and cb is equal to ab"    |

Note that Answer 3 is generated by using the common notion "things which are equal to the same thing are also equal to one another" following the meta-cognitive decision of the system that no statement in the proof can be used to complete the answer to the question.

## 7 CONCLUSION

The present paper presented a system for meta-cognitive question answering from texts as applied to the Euclid's "Elements" text. The system can answer "why" questions from geometric proof texts by representing and processing its own state of automatic proof analysis and generating explanations and supporting diagrams.

The system is implemented in Prolog and uses a question grammar combined with a text grammar. These two grammars use rules, lexicons, ontologies and the state of the system to answer the input questions. The questions are first analyzed into components and then are transformed in the cases where no direct match is found. Our system can be applied either to meta-cognitive tutoring [1] or to the testing of meta-cognitive mathematical reasoning models [8].

## REFERENCES

- [1] V. Alevan, et al "Toward Meta-cognitive Tutoring: A Method of Help-Seeking with a Cognitive Tutor". International Journal of Artificial Intelligence in Education, Vol 16, pp. 101-130, IOS Press, 2006.
- [2] Heath T. L. "The Thirteen books of Euclid's Elements", Dover N.Y. 1956.
- [3] J. Kontos "ARISTA: Knowledge Engineering with Scientific Texts". Information and Software Technology, Vol. 34, No 9, 611-616, 1992.
- [4] J. Kontos and I. Malagardi "Information Extraction and Knowledge Acquisition from Texts using Bilingual Question-Answering". Journal of Intelligent and Robotic Systems, Vol 26, No. 2, pp. 103-122, October, 1999.

- [5] J. Kontos, et al". "ARISTA Causal Knowledge Discovery from Texts". Proceedings of the 5th International Conference on Discovery Science DS 2002, Luebeck, Germany, pp. 348-355, 2002.
- [6] J. Kontos et al "The AROMA System for Intelligent Text Mining". HERMIS, International Journal of Computers mathematics and its Applications, Vol. 4, pp.163-173, LEA, 2003.
- [7] J. Kontos et al "Question Answering and Rhetoric Analysis of Biomedical Texts in the AROMA System". Proceedings of 7th Hellenic European Research on Computer Mathematics & its Applications Conference, Athens, <http://www.aueb.gr/pympe/hercma/proceedings2005>. 2005.
- [8] S. Shapiro et al "Metacognition in SnePS". AI Magazine, 28, 1, Spring Issue, 2007.