CocoQa: Question Answering for Coding Conventions over Knowledge Graphs

Tianjiao Du, Junming Cao, Qinyue Wu, Wei Li, Beijun Shen, Yuting Chen
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University, Shanghai, China
{tjsoulshe, junmingcao, wuqinyue, liwei, bjshen, chenyt}@sjtu.edu.cn

***Abstract*—Coding convention plays an important role in guaranteeing software quality. However, coding conventions are usually informally presented and inconvenient for programmers to use. In this paper, we present CocoQa, a system that answers programmer's questions about coding conventions. CocoQa answers questions by querying a knowledge graph for coding conventions. It employs 1) a *subgraph matching* algorithm that parses the question into a SPARQL query, and 2) a *machine comprehension* algorithm that uses an end-to-end neural network to detect answers from searched paragraphs. We have implemented CocoQa, and evaluated it on a coding convention QA dataset. The results show that CocoQa can answer questions about coding conventions precisely. In particular, CocoQa can achieve a precision of 82.92% and a recall of 91.10%.**
Repository: https://github.com/14dtj/CocoQa/
Video: https://youtu.be/VQaXi1WydAU

***Index Terms*—Coding convention, question answering, knowledge graph**

I. Introduction

Coding convention plays an important role in software development. Coding conventions are rules that programmers need to obey when programming; they vary from languages to languages, companies to companies, and even projects to projects. Coding conventions positively affects software development in that they do improve comprehensibility of the code, enhance quality, maintainability and reusability of a software product, etc. Meanwhile, coding conventions are typically trivial and informally presented, making it difficult for programmers to follow and/or check their products against conventions of interests.

For this reason, many efforts have been spent on developing facilities that allow programmers to program with coding conventions. One human-friendly solution is to develop question answering (QA) systems for coding conventions—a QA system allows programmers to interactively raise their questions and answers them.

Though automated question answering has been intensively studied, how to develop QA for coding conventions is still a challenging problem. So far there exist two mainstreams of QA systems: *knowledge base (KB)-based* QA systems and *corpus-based* ones.

A knowledge base can store complex structured data including relations and entities, and a KB-based QA system can query structured knowledge bases directly in order to answer a question [2], [5], [8], [10], [11], [15]. However, knowledge bases have inherent limitations. They are far from complete and usually lack of contextual information. Consequently, a question may not always have its answers in a knowledge base under querying.

A corpus-based QA system, on the contrary, searches and retrieves answers from unstructured corpus. A corpus-based QA system usually relies on *machine comprehension*, a technique that comprehends textual resources (i.e., documents and web pages on the Internet), to answer questions. Many deep learning techniques can be leveraged. For example, Wang et al. have adopted a match-LSTM model and the Pointer Net model to comprehend documents and generate answers to questions [14]. New training and evaluation datasets (e.g., SQuAD [9]) have also been released, making machine comprehension competent to human comprehension.

In this paper, we present CocoQa, a QA system for coding conventions. CocoQa combines the advantages of the KB- and the corpus-based QA systems: it stores in a knowledge base coding conventions collected from online resources; it builds CCBase, a knowledge graph introduced in [3], for modeling coding conventions; and it answers questions about coding conventions by querying the graph. CocoQa is publicly available both as a plugin of Intellij Idea[1], and as an online web service[2].

This paper makes two contributions:

1) **An approach to QA for coding conventions**. CocoQa integrates several techniques to perform QAs over CCBase. Given a question q, CocoQa answers it using: 1) a *subgraph matching* algorithm that casts q into a structured query (say sq) followed by querying CCBase, 2) a *machine comprehension* algorithm that applies an end-to-end deep neural network to answer sq on textual paragraphs, and 3) a *logistic regression classifier* trained to merge and rank the answers.
2) **Implementation and evaluation**. We have implemented CocoQa as an Intellij Idea's plugin and as well a web service, providing developers with supports in programming with code conventions. We have also evaluated CocoQa on a coding convention QA dataset. The results show that CocoQa can answer questions about

[1]https://github.com/14dtj/CocoQa/
[2]http://202.120.40.28:4463

978-1-7281-2508-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ASE.2019.00108

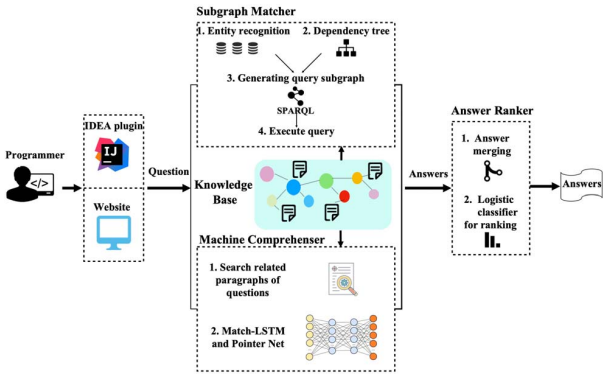

Fig. 1. An overview of CocoQa

coding conventions precisely. In particular, CocoQa can achieve a precision of 82.92% and a recall of 91.10%.

II. Design and Implementation of CocoQa

A. Overview

As Figure 1 shows, the CocoQa tool consists of four components: (1) a *coding convention knowledge graph* constructed from online resources, (2) a *subgraph matcher* that understands natural language questions and performs SPARQL queries over the knowledge graph, (3) a *machine comprehenser* that employs a deep neural network model to answer questions. The comprehenser searches answers from all textual paragraphs attached to the knowledge graph, and (4) a *ranker* that ranks answers retrieved by the subgraph matcher and the machine comprehenser via a logistic regression classifier.

B. Design

Next explain the four components of CocoQa.

Knowledge Graph. CCBase is a coding convention knowledge graph—structural data are stored in the form of entities and relations, and unstructured data as descriptive attributes.

CocoQa builds CCBase by following the approach presented in [3]. First, it designs the ontology of coding conventions, including rule, pros, cons, examples, exceptions, source code, and applications. Second, it crawls and collects data about coding conventions from books, standards organizations, open source organizations, universities, companies, etc. CocoQa recognizes entities and relations from the collected data and establishes the triples in the knowledge graph. CocoQa also uncovers and establishes the latent relations among entities, including similar-to, relate-to and subsumption relations.

We also collect unstructured data from online resources. The collected documents are split into paragraphs and titles and then attached to entities in CCBase. We use the TF-IDF weighted bag-of word vector to calculate the similarities between paragraphs and entities and attach a paragraph to an entity of the highest score.

The resulting knowledge graph contains 3,761 entities, 767 relations and 1,800 attached paragraphs. The graph can grow along with introducing new conventions.

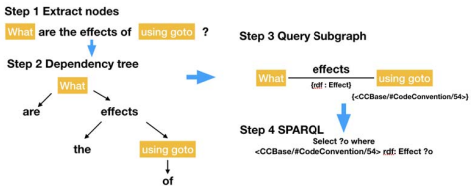

Fig. 2. A subgraph matching example

Subgraph Matcher. The subgraph matcher generates answers to a question by querying entities and their relations in CCBase. Given a question, it casts the question into a SPARQL query[3] followed by querying CCBase. Figure 2 illustrates how a question "*Q1. what are the effects of using goto?*" is transformed into a query.

More particularly, given a question, the subgraph matcher takes a subgraph matching algorithm [5] to extract entities and relations inside and mapping the question into a SPARQL query: (1) the matcher extracts nodes (including entities and wh-words) from the question. For example, in Figure 2, "*what*" (a wh-word) and "*using goto*" are identified as two nodes for *Q1*. Here Jena Full Text Search[4] is employed to identify entities in a question; (2) the matcher generates a dependency tree for *Q1*; (3) the matcher builds a graph Q_u: for each pair of nodes (v_i, v_j), if there is a path between them in the dependency tree, an edge is introduced in Q_u; the label is set as the concatenation of words along the path between v_i and v_j in the dependency tree. Q_u can be mapped to a subgraph of CCBase, containing only the nodes and edges w.r.t. the question; and (4) the matcher generates SPARQL queries from Q_u and finds candidate relations in CCBase for the edges.

Machine Comprehenser. The comprehenser generates answers by comprehending textual paragraphs attached as descriptive attributes of entities in CCBase. Thus the question answering problem becomes a multi-document comprehension problem.

Given a question, the machine comprehenser first searches the related paragraphs of the question. It extracts keywords from the question's representation. Similar to YodaQA [1], the comprehenser selects all noun phrases, noun tokens and the subjects (determined by a dependency parse) as the keywords. Then it uses the Apache's Solrsearch engine[5] to search full-text keywords in the texts and titles of paragraphs; the top 10 results are taken as potential paragraphs.

After that, the machine comprehenser employs Match-LSTM [13] and answer pointer network [12] for retrieving answers from the potential paragraphs inspired by S Wang's work [14]. This neural network consists of three layers. The first layer is an LSTM layer that preprocesses the paragraph and the question. The purpose is to encode the contextual information into each token of the paragraph and the question.

[3]https://jena.apache.org/tutorials/sparql.html
[4]https://jena.apache.org/documentation/query/text-query.html
[5]https://lucene.apache.org/solr/

The second layer is a match-LSTM layer using the attention mechanism to integrate the paragraph and the question. The third layer is an answer pointer layer that selects a set of tokens from the paragraph as the answer.

Let $\mathbf{H}^r$ be the hidden state in the last layer. The probability of generating an answer is

$$p(\mathbf{a}|\mathbf{H}^r) = p(a_s|\mathbf{H}^r)p(a_e|a_s, \mathbf{H}^r), \quad (1)$$

where a_s and a_e represents the start and the end position of the answer, receptively. The comprehenser globally searches for answers and retrieves those with highest probabilities, letting the probability of each answer be $p(a_s) \times p(a_e)$.

Note that *deep transfer learning* is used to train this network. We collected questions and answers from StackOverflow[6] to construct a coding convention QA dataset CocoQad, whilst the dataset is far from enough to train a deep network. Thus we adopt deep transfer learning to solve the cold start problem: we first train the model on some general purposed QA datasets like SQuAD [9] or MARCO [7], and then use our dataset to tune the pre-trained model. A detailed evaluation of the tuning will be given in the next section.

Answer Ranker. The answer ranker merges and ranks the answers obtained from the subgraph matcher and the machine comprehenser. A normalization process needs to be taken for removing duplicated answers.

Since ranking can be casted into a classification problem, we train a logistic regression classifier and sort the answers by their classification scores. The classifier learns the weights that allow correct answers to be distinguished from incorrect answers. During prediction, we rank candidate answers with confidence scores. The input to this classifier is a vector of numerical feature values—they are selected by human experts, including answer features (data source, search result score, etc.) and similarities of the answers to the question.

C. Implementation

We have implemented CocoQa, which can either be used as a plugin of Intellij Idea, or as an online web service. The plugin is implemented in Java, adopting JavaFX to embed a web view in the plugin panel. The web service is built in Python. Apache's Jena Fuseki is used as a SPARQL server. The CCBase is stored as RDF triples and can be accessed via a web service.

We adopted a python wrapper of Stanford CoreNLP [6] for generating the dependency trees of questions. As for the neural network in machine comprehenser, we employed a python implementation[7] of the Match-LSTM and answer pointer network. It makes some changes to the original network including replacing LSTM with GRU, adding a fully-connected layer, etc. The modification can increase the F1 score by 5.3% in the evaluation.

The snapshot of CocoQa is shown in Figure 3. In a chat dialog, programmers can raise their questions and the system answers these questions. The returned information is usually a paragraph containing highlighted answers. If the answers can be linked to the entities in CCBase, the attributes (pros, example, etc.) of these coding conventions are also accessible. Besides, the entities and their related entities are visualized in the form of force-directed graph.

[6]https://stackoverflow.com/

[7]https://github.com/laddie132/Match-LSTM

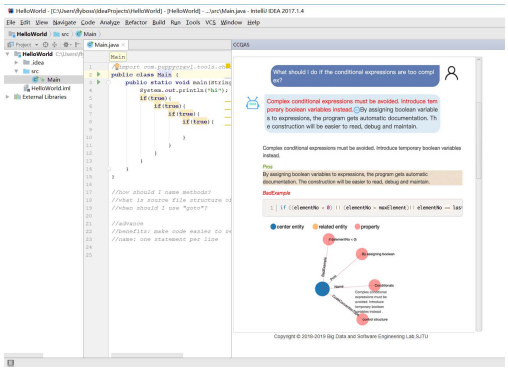

Fig. 3. A snapshot of the CocoQa plugin

III. Evaluation

We have evaluated CocoQa on our code conventions QA dataset. Our evaluation was designed to answer two research questions:

RQ1. Can CocoQa accurately answer questions about coding conventions?

RQ2. What is the best strategies of training Match-LSTM in machine comprehenser?

A. Setup

Dataset. We collected 214 real questions about coding convention from StackOverflow. These questions can be divided into two categories. One is to query attributes of coding conventions, and another is to query coding conventions related with conditions. For example, "*Is there any naming conventions of boolean methods?*".

We manually found the answers to the questions in CCBase. We also defined some question templates such as "*Tell me something about #name*"—values from the knowledge base can be retrieved to fill the variables and answers in the templates automatically.

The resulting coding convention QA dataset (say CocoQad) contains 1714 question answer pairs. This coding convention dataset is further divided into a training set (50%), a validation set (30%), and a test set (20%) in our evaluation.

Metrics. For measuring CocoQa's capability of question answering, we selected precision, recall, and F1 as metrics, which are often used to evaluate QA systems.

For measuring the training capability of Match-LSTM on CocoQad, we employed EM (Exact Match) and F1 as metrics, which were as well used for evaluating Match-LSTM on SQuAD.

TABLE I
A COMPARISON AMONG COCOQA

Component	Precision	Recall	F1
Subgraph Matcher	85.20	83.91	84.50
Machine Comprehenser	72.12	89.53	79.89
CocoQa	82.92	**91.10**	**86.82**

TABLE II
A COMPARISON OF STRATEGIES FOR TRAINING MATCH-LSTM.

		Dev		Test	
Strategy	**Training data**	EM	F1	EM	F1
Union	SQuAD + CocoQad	63.35	83.74	48.70	66.02
	MARCO + CocoQad	59.16	81.75	40.25	64.70
Tagging	SQuAD + CocoQad	61.78	86.60	51.30	66.43
	MARCO + CocoQad	60.21	83.65	44.15	66.69
Fine-tuning	SQuAD (training) CocoQad (tuning)	62.83	86.12	**53.24**	**72.53**
	MARCO (training) CocoQad (tuning)	56.64	81.45	39.61	66.30

B. Results to RQ1

We compared CocoQa against a QA system with subgraph matcher and another with machine comprehenser. As Table I shows, machine comprehenser can achieve higher recall but lower precision than subgraph matcher. One reason for this is that machine comprehenser learns knowledge from rich textual data and tends to return extra information.

As a combination of a subgraph matcher and a comprehenser, CocoQa can achieve high precision (82.92%), recall (91.10%), and F1 score (86.82%), indicating that CocoQa can answer questions about coding conventions precisely. Furthermore, answer ranker helps prioritize correct answers, which significantly facilitates programmers to obtain answers they need.

C. Results to RQ2

To solve the problem of insufficient training data, CocoQa takes a fine-tunning strategy: it transfers learning from some general-purposed QA datasets, like SQuAD and MARCO, and then tunes the model using CocoQad. We compare this strategy with two strategies:

1) A union strategy. We trained the machine comprehension model on a collection of CocoQad and SQuAD/MARCO.
2) A tagging strategy. This strategy is similar to the strategy designed by Chu et al. [4]. We appended tags ("*[2SQuAD]*" and "*[2CoCoQad]*", etc.) to data items to indicate their belongings. Besides, we oversampled CocoQad so that the training process can pay equal attention to each domain.

The results are shown in Table II. The union strategy is not satisfying, since CocoQad is too small to be comparable with the other datasets. The data items in CocoQad are also significantly different from those in the other sets, since the answers are usually complex, containing many programing language-specific terms and code snippets. The tagging strategy achieves little improvement over union strategy. Comparatively, the fine-tuning strategy achieves the highest EM and F1-score.

IV. CONCLUSION

This paper have presented CocoQa, a QA system that can answer questions for coding conventions over knowledge graph. CocoQa leverages several techniques (including subgraph matching and machine comprehension) so that precise answers can be retrieved. We have evaluated our system on a dataset that contains 1714 QA pairs. The evaluation results demonstrate the capability of CocoQa–it can achieve a precision of 82.92% and a recall of 91.10%, indicating that the answers are precise and do meet the programmers' needs.

ACKNOWLEDGMENT

This research was sponsored by the National Key Research and Development Program of China (Project No. 2018YFB1003903), National Nature Science Foundation of China (Grant No. 61472242 and 61572312).

REFERENCES

[1] P. Baudiš and J. Šedivỳ. Modeling of the question answering task in the yodaqa system. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 222–228. Springer, 2015.
[2] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.
[3] J. Cao, T. Du, B. Shen, W. Li, Q. Wu, and Y. Chen. Constructing a knowledge base of coding conventions from online resources (S). In *The 31th International Conference on Software Engineering and Knowledge Engineering*, 2019.
[4] C. Chu, R. Dabre, and S. Kurohashi. An empirical comparison of simple domain adaptation methods for neural machine translation. *arXiv preprint arXiv:1701.03214*, 2017.
[5] S. Hu, L. Zou, J. X. Yu, H. Wang, and D. Zhao. Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):824–837, 2017.
[6] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.
[7] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.
[8] Y. Qu, J. Liu, L. Kang, Q. Shi, and D. Ye. Question answering over freebase via attentive rnn with similarity matrix based cnn. *arXiv preprint arXiv:1804.03317*, 2018.
[9] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.
[10] U. Sawant, S. Garg, S. Chakrabarti, and G. Ramakrishnan. Neural architecture for question answering using a knowledge graph and web corpus. *Information Retrieval Journal*, pages 1–26, 2019.
[11] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.
[12] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.
[13] S. Wang and J. Jiang. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*, 2015.
[14] S. Wang and J. Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.
[15] W. Zheng, J. X. Yu, L. Zou, and H. Cheng. Question answering over knowledge graphs: question understanding via template decomposition. *Proceedings of the VLDB Endowment*, 11(11):1373–1386, 2018.