Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Semantic Parsing of Questions

Martin Matulík

Supervisor: Mgr. Petr Baudiš
Field of study: Computer and Information Science
Subfield: Natural Language Processing
May 2016

Acknowledgements

I would like to express my gratitude to my supervisor Mgr. Petr Baudiš for having patience with me and always giving me great advice for my work and to Ing. Jan Šedivý, Csc. for giving me a chance to get a hands on experience with machine learning and improve myself in that area. I would also like to thank RNDr. Petr Olšák and Ing. Tomáš Hejda for creating a LaTeXtemplate making my work writing this document much easier.
Last but not least I would like to thank my family for supporting me through my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 26. May 2016

....

Martin Matulík

Abstract

We examine Xser, a system answering questions over linked data. It works in two parts. First, it utilizes a shift-reduce parser to form a structure representing the question. Then, the structure can be linked to a knowledge base and converted to a query in structured language. We implement the system and perform experiments to evaluate our work.

Keywords: semantic parsing, shift-reduce, question answering, natural language processing

Supervisor: Mgr. Petr Baudiš
ČVUT Media Lab,
BLOX,
Evropská 11
160 00 Praha 6

Abstrakt

V této práci zkoumám Xser, systém pro odpovídání na otázky čerpající ze strukturovaných dat. Tento systém funguje ve dvou částech. Nejdříve použije shift-reduce parser pro sestrojení struktury reprezentující otázku. Poté tuto strukturu propojí s grafovou databází a převede na dotaz ve strukturovaném jazyce. Má práce sestává z implementace systému Xser a provedení experimentů.

Klíčová slova: sémantické parsování, odpovídání na otázky, zpracování přirozeného jazyka

Překlad názvu: Sémantické parsování otázek

Contents

Figures

Tables

Chapter 1
Introduction

The need to store general knowledge in a structured way gave birth to several large graph databases such as Freebase[BTPC07]. A task in semantic parsing that drawn a lot of attention recently is understanding natural language question and retrieving answer to it from database. Two tasks need to be solved: recognizing predicate-argument structures in the question and linking it to given knowledge base.
Most semantic parsers aim to solve these tasks at the same time which, however, takes way too long (up to days)[BCFL13] to train due to large sizes of knowledge databases. On top of that, they usually cannot be reworked to reference different knowledge bases. Xser [XFZ14] is a new system that approaches the problem in a slightly different way. Its authors assume that intention of the question can be recognized independently on any knowledge base and only when the structured intention is known the structure is linked to given knowledge base. We examine this system by implementing it in Python language. We then use our implementation for trying to reproduce results on various tasks by performing our experiments.
The thesis is structured as follows: First part is a theoretical background composing of chapters 2 and 3. In chapter 2, we survey various approaches to semantic parsing and in chapter 3 we explain how a shift-reduce parser works. Second part describes our implementation of the Xser system with chapter 4 about algorithms and chapter 5 documenting our code. The last part covers our experiments with the system. Chapter 6 and chapter 7 contains results of our tests.

1.1 Used software

The programming was done in Python language using PyCharm IDE[1]. Whole project code is available on enclosed CD. All pictures appearing in this document were done by us using Inkscape[2] graphic editor. The thesis was written in LaTeXwith Overleaf web service[3].

[1]www.jetbrains.com/pycharm/
[2]www.inkscape.org/
[3]www.overleaf.com

Part I

Survey

Chapter 2
Semantic parsing

The term *semantic parsing* covers wide area of tasks. In this chapter we focus on the most common definition: mapping a natural language sentence to its representation in formal language (e.g. lambda calculus). We present various approaches to this problem.

2.1 Approaches

2.1.1 CHILL

CHILL parser input acquisition system was created by [ZM96]. Its original purpose was finding a set of control rules for a shift-reduce parser from a set of natural language questions and their respective parses [Zel95]. As the system is written in Prolog, inductive logic programming is used for finding the control rules. The resulting parser takes a sentence on input and returns logic clause for this sentence.
It was later modified to perform parsing questions to a structured query language, in this case GeoQuery. The modification lies in replacing the shift-reduce parser actions with new actions $introduce$, $co-reference$ and $conjoin$. The resulting parser is then applied on a question and structured query is returned.

2.1.2 PCCG

Probabilistic combinatory categorial grammars (PCCGs) are used in the GENLEX system [ZC05] and in the work of [KGZS12].
A CCG can parse a sentence to one or more logical forms. It is built around a lexicon Λ composed of pairs (*word*, *syntactic type*). The syntactic types could be primitive (entities) or complex (relations). Semantic type is usually also added to the lexicon. Furthermore, the CCG has a set of rules for recursive combination of the syntactic types.
More than one parse is usually found under rules of a CCG. PCCG solves this problem by finding the probabilities of all parses using a log-linear model then choosing the best parse.
GENLEX estimates the parameters using dynamic programming. For training

and evaluation it uses Geo880 and Job640 datasets both split into training and testing splits. Kwiatkowski trains the model on the Eve corpus with online Variational Bayes Expectation Maximization.

2.1.3 MATCHER, LEXTENDER

Paper of [CY13] focuses on schema matching and pattern learning. It composes of two parts: MATCHER and LEXTENDER. The authors developed the $Free917$ dataset which our framework utilizes.
The task of MATCHER is finding correct matching for natural language utterance given a textual schema and a database schema.First, candidates are generated using Web search engine query. Pattern matching rules are learned using further web queries. Statistics of the patterns are also collected during this steps and used when training the regression model for scoring the candidates. Lastly, best candidate according to the model is chosen.
The framework is again based on PCCG. The task of adding semantic types into PCCG lexicon Λ is delegated to the LEXTENDER system. It tries to predict syntactic type, semantic type and weight W for each relation in matching output from the MATCHER. Models for syntactic and semantic types are Naive Bayes. Features for syntactic type include POS tags and information about the relation. For semantic type features, predicted syntactic type is added. Model for finding weights is linear regression.

2.1.4 Semantic parsing from Freebase

[BCFL13] introduce a parser based on a set of compositional rules and lexicon mapping phrases in natural language to knowledge base predicates and entities. This work differs from previous ones by defining only simple rules then learning finer rules and conditions.
The authors constructed new dataset of question-answer pairs called WEBQUESTIONS which they used for training and evaluation. In separate experiment they also utilize the $Free917$ dataset.

2.1.5 PARASEMPRE

PARASEMPRE from [BL14] works in three steps. First, it generates a set of logical form candidates for given utterance. Next, it forms a list of canonical utterances for each predicate from knowledge base. Last step is choosing a canonical utterance which paraphrases the input utterance the best. The model tries to maximize the log-likelihood of a correct answer for the question. As the authors are the same as in previous system, $Free917$ and WEBQUESTIONS dataset are used for training and evaluation.

Chapter 3

Shift-reduce parsing

3.1 Directed acyclic graphs

Data-driven natural language dependency parsing algorithms used in information extraction require the sentence structure to be represented as a tree with words as nodes and head-dependent semantic relations as labelled edges. Despite advantages such as computation efficiency this representation has its limitations: relations deeper than shallow syntax cannot be captured by trees. As opposed to trees, directed acyclic graphs (DAGs), can capture long-distance relationships and other deeper dependencies and linguistic phenomena while remaining computationally efficient.[ST08] In dependency tree, each word has only one head. The DAG, however, allows for each node to have multiple heads.

3.2 Approaches to parsing a DAG

There are multiple approaches to parsing a DAG from a sentence. One of them is creating a tree out of the question and adding edges to form a DAG. Another one is shift-reduce transition-based parser [ST08]. Because parsing the question with this parser is based on local decisions and rich set of features, the computational cost is quite low. The disadvantage of shift-reduce parser is that it is unable to output non-projective graph structures.

Definition 3.1. [NN05] We use $w_i \to w_j$ to denote there exists an unlabelled arc between w_i and w_j, $w_i < w_j$ to denote w_i precedes w_j and $w_i \to^* w_j$ to denote the arc is closed on reflexivity and transitivity.

1. An arc $w_i \to w_k$ is projective $\iff \forall w_j, w_i < w_j < w_k$ it holds that $w_i \to^* w_j$
2. A dependency graph $D = (W, A)$ is projective $\iff \forall a \in A$ it holds that a is projective.

To put it informally, projective graph has all edges above nodes and no edge cross each other. To counter this limitation, DAG could be modified by pseudo-projectivization techniques (see subsection 3.5.1).

3.3 Parser state

The parser state consists of three elements: queue, stack and current DAG configuration (in bottom-right corner of figure 3.1 and following figures). Queue is initialized the sequence of input tokens as it is formed in previous step from the text of the question. Stack starts empty. DAG is initialized to tokens as nodes with no dependency edges between them. Tokens are pushed onto the stack and arcs are formed between the top of the stack and the head of the queue which is then reflected in the DAG.

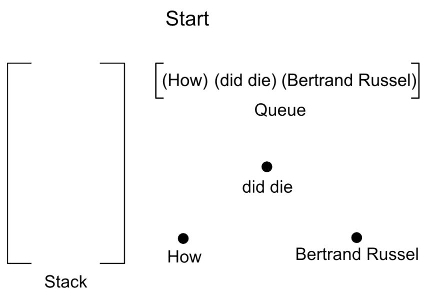

Figure 3.1: Starting state of parser for question 'How did Bertrand Russell die?'

3.4 Parser actions

The shift-reduce parser performs in each state one of its available actions. This is non-deterministic since multiple actions could be taken in most states. The parser needs to be guided in order to perform correctly.

3.4.1 Shift

First possible action is shift. By shifting the parser removes the head of the queue from the input queue and pushes it onto the stack. Next item of the queue becomes the new head.

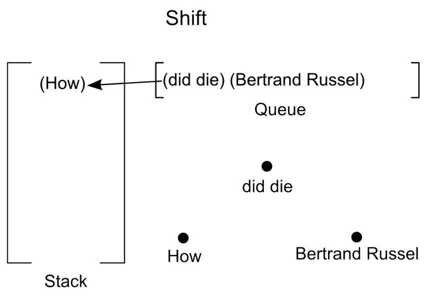

Figure 3.2: Shift action

3.4.2 Reduce

Reduce action pops the top of the stack and discards it altogether from the parsing process. In the standard dependency parser, reduce action is taken only when the stack top already has at least one head. This ensures the resulting graph is connected. In our parser, however, we are interested only in dependencies between predicate and argument and thus the graph could be left unconnected.

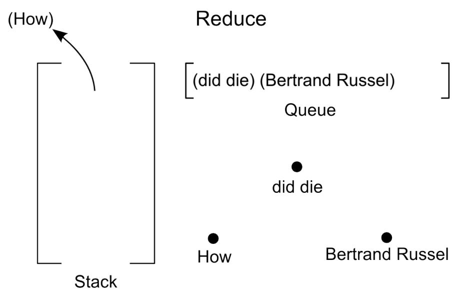

Figure 3.3: Reduce action

3.4.3 Arc-left, arc-right

Actions forming arcs between tokens are left-arc and right-arc. They are performed only in case no arc already exists between the tokens. Left-arc creates a dependency arc between the top of the stack to the head of the queue where the queue item is the head and the stack top is the dependent. Right-arc action forms an arc where the queue item is the dependent and the stack top is the head of the dependency. There is no change in stack nor queue in left-arc and right-arc parser actions. The parser allows for a token to have multiple dependants as well as multiple heads.

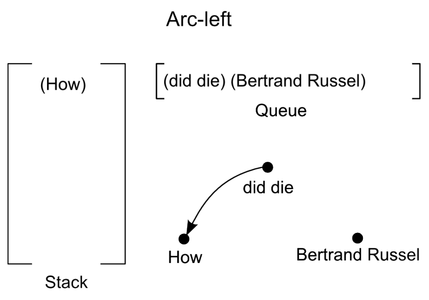

Figure 3.4: Arc-left action

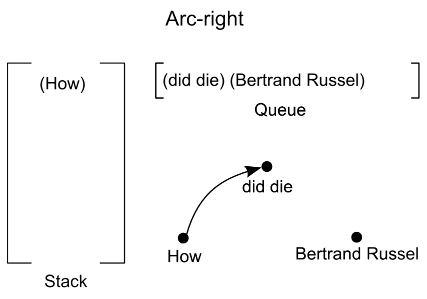

Figure 3.5: Arc-right action

3.5 Modifications of shift-reduce parser

3.5.1 Pseudo-projectivization

[NN05] suggest a way to pseudo-projectivize the structures output by a shift-reduce parser. First, training data structures are pseudo-projectivized using minimal number of lifting operations and new edges are marked. Lifting operation is applied on edge that fails the condition of projectivity and replaces its head node w_j with node w_i, which is the head of w_j.
Next, shift-reduce parser is trained and when finished it should be able to assign special labels to edges that ensure the projectivity. Finally, structure is transformed using the marks on edges and a set of rules.

3.5.2 Right-reduce, left-reduce

Two actions replacing left- and right-arc are used in [Niv04]. Instead of connecting stack top and head of the queue, these actions connect top two items of the stack with an arc of corresponding direction (left-/right-) and remove the top item from stack (reduce).
This ensures the structure being built during the parsing process is connected at all times.

Part II

Implementation

Chapter 4
Algorithms and approaches

4.1 Structured perceptron

Modification of perceptron aimed for sequence tagging is described in [Col02]. It combines this classic algorithm with an inference algorithm such as Viterbi algorithm to be used in a tagging task. The task is defined as correctly assigning a sequence of tags (e.g. part-of-speech tags) $y_1, y_2, \cdots, y_n$ to a sequence of samples (e.g. words) $x_1, x_2, \cdots, x_n$, where n is the length of the sequence.
Each pair (training sample, label) is mapped by a feature function Φ to a vector of fixed length. Feature in part-of-speech tagging task could be, for example,

$$\phi_{100}(x, y) = \begin{cases} 1 & \text{if } x = \text{is}, y = \text{V} \\ 0, & \text{otherwise} \end{cases}$$

where V is part-of-speech tag for verb. Other features could be bigrams and trigrams of words and tags. They are constructed from a vocabulary of words and tags the same way as the example feature. This results in long sparse feature vectors for each example.
Parameter vector α of the same length as the feature vectors is initialized to zero vector. Viterbi algorithm generates a set of tagged sequence candidates denoted Y and classifier predicts the best one of them subject to

$$y^* = arg \max_{y \in Y} \Phi(x, y) \cdot \alpha$$

If the predicted sequence is different from the gold standard sequence, weight of incorrect parameters is decreased by value of the learning rate and the weight of correct parameters is increased. If the predicted sequence is correct, no change to parameters happens.

4.2 Part-of-speech tagging

In the process of part-of-speech (POS) tagging, each word in a corpus is marked with a tag corresponding to its role as a part of speech. The role is defined by its relationships with adjacent and relative words in a sentence or

paragraph. There are 9 main parts of speech in English, but these are further split into more finely distinguished categories based on the gender, number or case of the word. The task of part-of-speech tagging is not easy due to the fact a single word could represent different part of speech in different sentences.
In our work we use tagset from Penn Treebank Project [San04] provided by the nltk Python library.

4.3 Named entity recognition tagging

Named entity recognition (NER) is the task of extraction of information from text [NS07]. It requires telling names and numbers apart from other words in text and then correct classification into one of several categories: people, companies, places, years etc.

4.4 Xser system overview

The Xser system [XFZ14] works in four steps: First, phrases are detected in the processed question. The phrases are then parsed with a shift-reduce parser to form a DAG with them as nodes. That concludes the part which is independent to any knowledge base. Next, the DAG is linked to a given knowledge base predicates and arguments. Finally, a query in structured language is constructed from the linked DAG and sent to retrieve answer. More detailed description of each step follows.

4.5 Phrase detection

The first phase of parsing is phrase detection. In the original work, each word in a question is assigned a label from the set {*entity*, *relation*, *category*, *variable*, *unlabeled*}. Figure 4.1 shows an example of labelled question.

$$How_{variable}\ did_{relation}\ Bertrand_{entity}\ Russell_{entity}\ die_{relation}?$$

Figure 4.1: Example of a question with labelled phrases

Variable is interrogative pronoun or pro-adverb of the question, for example who or when. Entities are usually subjects of the question, like people or items. They correspond to knowledge base classes.
Words labelled as category represent the property of given entity which the asker wants to know about. In our work, however, we dropped the category label and moved the words originally labelled as category into the “relation” set. This was done because edges from category token created non-projective graphs which the shift-reduce parser cannot handle. On top of that, not all question had word or phrase fulfilling the role of "category". Instead, these

n	Description	Templates
1	unigram of POS tag	p_i
2	bigram of POS tag	$p_{i-1}\ p_i$,$p_i\ p_{i+1}$
3	trigram of POS tag	$p_{i-2}\ p_{i-1}\ p_i$, $p_{i-1}\ p_i\ p_{i+1}$, $p_i\ p_{i+1}\ p_{i+2}$
4	unigram of NER tag	n
5	bigram of NER tag	$n_{i-1}\ n_i$,$n_i\ n_{i+1}$
6	trigram of NER tag	$n_{i-2}\ n_{i-1}\ n_i$, $n_{i-1}\ n_i\ n_{i+1}$, $n_i\ n_{i+1}\ n_{i+2}$
7	unigram of word	w_i
8	bigram of word	$w_{i-1}\ w_i$,$w_i\ w_{i+1}$
9	trigram of word	$w_{i-2}\ w_{i-1}\ w_i$, $w_{i-1}\ w_i\ w_{i+1}$, $w_i\ w_{i+1}\ w_{i+2}$
10	previous phrase type	t_{i-1}
11	previous phrase type and current word	$t_{i-1}\ w_i$

Table 4.1: Features for phrase detection

words and phrases were moved to "relation" group where the framework can profit from extra information extracted from them.
Words describing the relation between entity and its property are labelled as such. They are mapped to knowledge base predicates.
The phrase detector marks words like articles (unless they are part of an entity name) or in certain cases prepositions for deletion by tagging them as "unlabelled". The problem can be formulated as sequence labelling task with the question text on input and sequence of tags on output.
The tagging is done via a structured perceptron. Table 4.1 describes used features. POS stands for part-of-speech tags, NER stands for Named Entity Recognition. After successful labelling of all words in a question, words are grouped by their respective labels and formed into tokens. Unlabelled words are dropped and the parser does not work with them any further. While there may be only one token of the entity, category and relation kind, it is possible to have multiple entities in a single question.

4.6 Shift-reduce parsing

Algorithm that performs the DAG decoding is depth-first beam search. First item is initialized with empty stack and question converted to list of tokens as a queue. This starting item is put on agenda. Parser then takes out items and applies all possible cases to them. In some cases, the reduce action (due to empty stack) or left/right arc action (due to already present edge) cannot be taken. All items newly created from the original item are inserted into the agenda. To make the computation more efficient, all items are scored when inserted into agenda and only the K best for fixed K are then chosen for further processing; the rest of the agenda is discarded. If the item on the agenda has no items in the queue, it is considered as finished. This item's score is then compared to output candidate score, and the output candidate

Category	Description	Templates
lexical	stack top	STw; STp; $STwp$; ST_e
features	queue head	N_0w; N_0p; N_0wp
	next phrase in queue	N_1w; N_1p; N_1wp
	ST and N0	$STptN_0pt$; $STptN_0t$
	POS bigrams	N_0pN_1p
	POS trigrams	$N_0pN_1pN_2p$
	N0 phrase	$N_0wN_1pN_2p$
semantic	Conjunction of	N_0t; N_0wt; N_0pt;
features	phrase label	N_1t; N_1pt; $STpN_0t$;
	and POS tag	$STtN_0p$; $STwN_0t$;
		$STpN_0p$; $STtN_0t$
structural	Arc-left exists	$AL(STt, N_0t)$
features	Arc-right exists	$AR(STt, N_0t)$;

Table 4.2: Features for shift-reduce parser training

is replaced in case of better score. When no items remain on the agenda, the shift-reduce parsing finishes and the DAG of candidate output item is returned as DAG representing the question.
Since the parser is non-deterministic, it needs an oracle to choose the next action. For that, we use the structured perceptron again. Training of the model is done in the following way: First, shortest sequence of actions leading to correct DAG is found by performing a depth-first search on the question. The script automatically creates features for the parser in its every state so the sequence of features is saved as well. Pairs $(feature\_vector, correct\_action)$ can then be created and model can be trained on them the same way as for phrase detection. At parsing time, parser in each state chooses the best action according to the model.
Table 4.2 describes features used in shift-reduce parser training. ST stands for stack top, N_i stands for i-th item in queue, w is phrase text, p is phrase POS tag and t is phrase label of the item.
We followed the original Xser paper when implementing this part.

4.7 Knowledge base linking

Linking entities to knowledge base classes and relations or DAG edges to knowledge base predicates is done in several steps. All of the approaches differ from the original paper and were chosen by us.

4.7.1 Entity linking

Class candidates for entity linking are obtained from Freebase endpoint using Google API. Only fixed number of them is kept for each entity. Features are then constructed based on candidate popularity, word coincidence and human-readable ID coincidence. Classification task with logistic regression as training model is performed on the data and best candidate is chosen as entity representing the token.

4.7.2 Relation linking

Relation linking is based on the bag-of-words model. Vocabulary of all words labelled as entity is stored and features are phrases encoded into one-hot vectors based on this vocabulary. Additionally, type of question (recognized by the interrogative pronoun) is taken into consideration when creating the features. Set of all predicates appearing in the whole dataset as a label to relation phrases is set as the output to the classification task. It is possible for SPARQL variable to appear as a label for relation phrase in a question with multiple entities. Classification is done via SVM.

4.7.3 Edge linking

Edges of the DAG are linked either to knowledge base predicates (in case of question with multiple entities) or to either of the tags SP,PO (in case of simpler question). SP is linked between entity phrase and relation phrase and means that the entity phrase is the subject of the relation. PO stands for entity being object of the relation.
Model for classification is bag-of-words again, this time the words of edge target and of edge head are examined. Type of question (recognized by variable) is also considered in the features.

4.8 SPARQL language conversion

Conversion from a linked DAG to a SPARQL query is done through a fixed set of conversion rules, same as in the original paper. RDF conditions are constructed from triples $(node, edge, node)$. Type of question is reflected during this phase. Figures 4.2 and 4.3 describe two of the rules used.

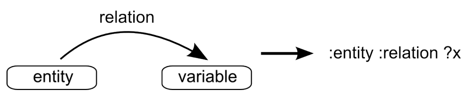

Figure 4.2: Example of conversion rule: edge is labeled as relation.

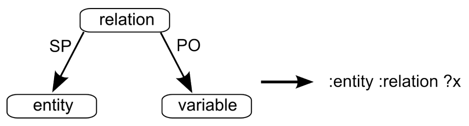

Figure 4.3: Example of conversion rule: relation refers to both entity and variable. SP and PO tags could be switched; In that case, entity and variable would also be switched in the resulting query.

Chapter 5
Code documentation

5.1 Used libraries

Our work was done almost entirely in the Python language with one Java library being an exception.
Data were loaded from json format using json library. Files created when running various parts of the framework such as processed text or trained models were stored in pickle format.
For part-of-speech tagging, POS tagger from nltk[1] library was used. NER tagging was done using Stanford NLP tagger[FGM05] in Java.
In linking part, scikit-learn[2] library was used for classification training and prediction. Feature processing methods in scikit-learn were also used for converting features to one-hot array, where numpy library was required.
Candidates for entities were obtained using Google Freebase API. Answers to constructed SPARQL queries were sent to a Freebase endpoint via SPARQLWrapper library methods.

5.2 System overview

Figure 5.1 explains how the question is processed by the system. Detailed description of each script follows.

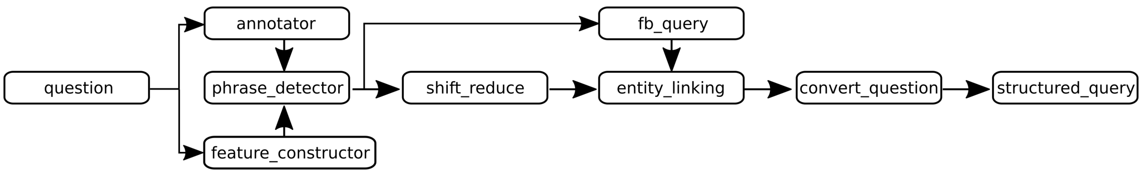

Figure 5.1: Illustration of how the question is processed through the pipeline

5.3 Phrase detection

Files used in detecting phrase type of a word.

[1]http://www.nltk.org/
[2]http://scikit-learn.org/

- **feature_constructor** - creates feature vectors of words based on their POS and NER tags. Phrase label of previous word and information about adjacent words in the question are also represented in the features. The feature vector has the form of list of *template_value* (see section 4.2 for the templates) strings in order to avoid sparse numeric vectors.

- **phrase_detector** - trains a model for phrase detection using the structured perceptron algorithm. The model has shape of hashmap (dictionary) where keys are unique features (*template_value* again) and values are dictionaries with classes as keys and actual weights as values. During training time, script extracts small dictionaries from the big dictionary using the feature vector from **feature_constructor** and sums the weight values for each class. Then it predicts the class with highest score as the candidate class. If the candidate class is different from the correct class, all weights from smaller dictionaries for the incorrect class are decreased by learning rate and for correct class they are increased by the same value.

5.4 Shift-reduce parsing

- **shift_reduce** - contains the methods for training the model, performing the shift-reduce parsing on one or more question and for evaluating the model.

5.5 Knowledge base linking

- **fb_query** - inputs a query for Freebase entity through Google Freebase API. Query about entity properties is also supported both through Google Freebase API and SPARQL.

- **entity_linking** - performs linking of DAG edges and nodes to knowledge base entities and relations.

5.6 SPARQL language conversion

- **structured_query** - takes a DAG linked to knowledge base and converts it to a set of SPARQL WHERE conditions. Then it forms whole SPARQL query and either sends it to a Freebase endpoint or saves it to a file.

- **convert_question** - processes a question through the whole pipeline: detects phrases, parses a DAG, links to knowledge base, converts to SPARQL query, sends the query to a Freebase endpoint and analyses the result. All models have to be trained before running. Question can be taken from the dataset or input as string.

5.7 Utility scripts

- **annotator** - this script allows the user to create gold standard for phrase labels. This is the only task where the labels cannot be learned by machine processing and have to be input manually. Only way to partially avoid this is to use bootstrapping which is also implemented here. It takes a subset of training data tagged by user and trains a model on them. Then the script uses the model to label small part of the unlabelled data and adds them to the training set. The steps are repeated until all data is labelled.
- **feature_processor** - converts feature vectors from list of strings representation to sparse one-hot vector representation. It utilizes methods from scikit-learn and numpy libraries.
- **answer_convertor** - finds out which questions cannot be answered from Freebase and inserts gaps where necessary so for each question, answer is on the same line. Originally the answer data were stored without gaps.

Part III

Experiments

Chapter 6
Data

6.1 Freebase

Freebase [BTPC07] was created in 2007 with the purpose of storing data in structured way and allowing collaborative extension of the knowledge graph. Freebase data are structured into triples in the RDF format. RDF is a directed, labelled graph data format for representing information in the Web. The data is obtained using SPARQL query language.

6.2 SPARQL query language

SPARQL language [1] is a semantic query language for databases which store data in RDF format. Query in SPARQL language consists of two parts: the SELECT clause which specifies the variables (recognized by starting with question mark symbol) to be present in the result and the WHERE clause which provides a basic graph pattern to be matched against the queried graph. Solution is the found by binding variables to RDF terms. All conditions given in WHERE clause have to be satisfied for result to be returned. They are written in the pattern of whitespace-separated list of subject-predicate-object triples. Further restrictions can be applied using the FILTER keyword. FILTER allows for example limit numerical values or filter out responses in different languages. Database for relations and entities and standard for datatypes can be specified as a prefix using the PREFIX keyword.

6.3 Free917 dataset

Free917 dataset [CY13] composes of 917 questions. Each question is represented by natural language formulation (called “utterance”) and a structured query in lambda calculus (called “target formula”). On average, a question is 6.3 words long. There are 81 distinct domains covered in the questions and the logic formulas contain 635 different Freebase relations. The dataset was created by two native English speakers without any restrictions on their

[1]https://www.w3.org/TR/rdf-sparql-query/

domains except that the domains should be diverse enough. 884 out of the 917 questions are actually answerable by Freebase based on gold standard answers[2]. The dataset is split into 641 training set and 276 questions for evaluation.
Following modifications were made to the data by us. Certain names of entities such as people's names or company names had their first letters initialized to help with NER tagging. Several typos were fixed in order to improve training of entity linking. Parentheses in certain original logic formulas had spaces between them which impeded parsing the formula into gold standard for DAG, entities and edges so the spaces had to be removed.

[2]Obtained from https://github.com/pks/rebol

Chapter 7
Results

7.1 Sub-results of all parts

Authors of the original Xser system [XFZ14] do not mention accuracy of each part so we did not have comparison for our results.

7.1.1 Phrase detection

The models were trained with 50 iterations over the training data as this gave the best results in broader experiments.

Learning rate	**Accuracy**
0.01	90.7%
0.1	91.4%
1	91.3%
10	91.5%

Table 7.1: Overall accuracy of phrase detection model

7.1.2 Shift-reduce parsing

Various sizes of beam were tested, ranging from 10 to 500, but no change had any impact on the result. We examined what impact had number of iterations of model training on the accuracy. The accuracy is number of correctly formed DAGs divided by total number of questions from training set.

Number of iterations	Accuracy
5	59.8%
10	32.9%
20	62.3%
50	55.0%
100	54.7%
200	53.3%
500	54.3%

Table 7.2: Overall accuracy of shift-reduce parser model

The problem with training and running a shift-reduce parser is that when an incorrect decision is made, the parser is not likely to recover from this mistake and form the correct DAG, even though all other actions were correct.

7.1.3 Knowledge base linking

Entity	Relation	Edge
87.4%	31.5%	74.9%

Table 7.3: Accuracy of each of the parts of knowledge base linking

The bag-of-words model does not work well for the relation labelling task. This is because the same words do not represent the same property in different questions. For example, one question in training data asks about a team certain basketball coach manages with words *What team does <person> coach?*. In the testing set, there is the same question but given the different coach's name, the team is supposed to be an ice hockey team. The knowledge base properties are different for each sport. The relation gets linked to basketball team instead of hockey team since the bag-of-words model cannot capture the difference. Same error could happen when question asks about time a discovery was made but the query retrieves the person who made it.
On the other hand, if the question resembles a question in the training dataset, the system is very likely to retrieve a correct answer. Table 7.4 mentions several relations that the system is able to correctly recognize and answer the questions that contains them.

Text	Representation of relation
When was Bill Clinton born?	people.person.date_of_birth
What is the area of the Czech Republic?	location.location.area
What is the origin of the Labe river?	geography.river.origin
Who directed Fight Club?	film.film.directed_by

Table 7.4: Examples of questions the system is able to answer

There are limitations to the system such as that the names have to be capitalized so the phrase detection correctly classifies them as entities.

7.2 Overall results

Our system was able to correctly answer only 29 out of 276 testing questions, making its accuracy 10.5%. This is to be expected if we examine the accuracies of separate parts of the system.
As illustrated by table 7.4, the system performs well on simple questions. The training dataset, however, composes of both simple and more complex questions with multiple entities and hard to capture relations. Sometimes the data is structured in the database so that the value of property is not available directly from entity but needs to be extracted from extra node. We were able to cover this in the gold standard for separate model, but the classifiers were unable to handle this structure well.
When we trained all parts of the system on simpler training data including DAGs with only three nodes and two relations (which is the form of most questions anyway), we were able to achieve accuracy of 24.6% (68 out of 276 questions correctly answered) on the testing data. This result came close to original results of Cai and Yates[CY13] who achieved 26.9% baseline accuracy on the same data. However, this improvement came at the cost of universality of the system.

Chapter 8
Conclusions and future work

We attempted to recreate the Xser system and reproduce its results. The best result we were able to achieve on the testing data was 24.9% while the current state is able to answer 10.5% of the questions. The main reason of the poor results is problematic linking of the relation phrases to knowledge base predicates.
On the other hand, there is almost no problem with answering simple questions since all other components work as expected. Also, the system is universal enough to be adapted to a different knowledge base than Freebase.
The next step to improve our system would be changing a model for relation labelling which is the weakest point of the pipeline. Models for shift-reduce parsing and edge labelling also need revising. Another goal is incorporating our work into YodaQA [1], a question answering system built by ailao composing of various frameworks.

[1] http://ailao.eu/yodaqa/

Appendices

Appendix A
Program demonstration

If you wish to test the program, you need to have Java and Stanford NLP package installed in addition to the packages mentioned in section 5.1. Then, run the **convert_question** script from question_conversion package with the following parameters, in this order:

- path to where you extracted the code (string)
- 276 - size of the testing dataset
- 50 - part of name of the model for phrase detection
- 20 - part of name of the model for shift-reduce parsing
- tst - testing mode (string)
- i - mode where user inputs questions for answering (string)
- path to your Java installation (string)
- path to your Stanford NLP installation (string)

Internet connection is also required to retrieve an answer from Freebase. For more options and instructions on running various parts of the pipeline please refer to the README file of the project. The system has been tested to run on Windows operating system but should be optimised to run on any system.

Appendix B

Bibliography

[BCFL13] J. Berant, A. Chou, R. Frostig, and P. Liang, *Semantic Parsing on Freebase from Question-Answer Pairs*, Empirical Methods in Natural Language Processing (2013).

[BL14] J. Berant and P. Liang, *Semantic Parsing via Paraphrasing*, Association for Computational Linguistics (ACL) (2014).

[BTPC07] K. Bollacker, P. Tufts, T. Pierce, and R. Cook, *A Platform for Scalable, Collaborative, Structured Information Integration.*

[Col02] M. Collins, *Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms*, EMNLP (2002).

[CY13] Q. Cai and A. Yates, *Large-scale Semantic Parsing via Schema Matching and Lexicon Extensions*, ACL (2013).

[FGM05] J. R. Finkel, T. Grenager, and Ch. Manning, *Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling*, ACL (2005), 363–370.

[KGZS12] T. Kwiatkowski, S. Goldwater, L. S. Zettlemoyer, and M. Steedman, *A Probabilistic Model of Syntactic and Semantic Acquisition from Child-Directed Utterances and their Meanings*, EACL (2012), 234–244.

[Niv04] J. Nivre, *Inductive Dependency Parsing*, Tech. Report MSI 04070, Växjö University, School of Mathematics and Systems Engineering, 2004.

[NN05] J. Nivre and J. Nilsson, *Pseudo-Projective Dependency Parsing*, 99–106.

[NS07] D. Nadeau and S. Sekine, *A Survey of Named Entity Recognition and Classification*, Lingvisticæ Investigationes **30** (2007), 3–26.

[San04] B. Santorini, *Part-of-speech tagging guidelines for the Penn Treebank Project*, Tech. Report MS-CIS-90-47, Department of Computer and Information Science, University of Pennsylvania, 2004.

[ST08] K. Sagae and J. Tsujii, *Shift-reduce Dependency Dag Parsing*, Proceedings of the 22nd International Conference on Computational Linguistics (COLING) **1** (2008), 753–760.

[XFZ14] K. Xu, Y. Feng, and D. Zhao, *Answering Natural Language Questions via Phrasal Semantic Parsing*, CEUR Workshop Proceedings **1180** (2014), 1260–1274.

[ZC05] L. S. Zettlemoyer and M. Collins, *Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars*, Proceedings of the Conference on Uncertainty in Artificial Intelligence (2005).

[Zel95] J. Zelle, *Using Inductive Logic Programming to Automate the Construction of Natural Language Parsers*, Ph.D. Thesis at University of Texas at Austin (1995).

[ZM96] J. Zelle and R. Mooney, *Learning to Parse Database Queries using Inductive Logic Programming*, Proceedings of the Thirteenth National Conference on Aritificial Intelligence (1996), 1050–1055.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	Martin M a t u l í k
Study programme:	Open Informatics
Specialisation:	Computer and Information Science
Title of Bachelor Project:	Semantic Parsing of Questions

Guidelines:

Investigate the problem of determining a formal representation of a given question by parsing it into a constraint query.

The student will survey the current methods of semantic parsing and implement a state-of-art approach (like [1]) within the YodaQA platform. The impact of the algorithm on the question answering pipeline shall be measured and reuslts of the evaluation discussed in the context of past work within semantic parsing.

Bibliography/Sources:

[1] Xu, K., Y.Feng, and D. Zhao. "QALD-4: Answering Natural Language Questions via Phrasal Semantic Parsing." 2014.
[2] Berant, Jonathan, and Percy Liang. "Semantic parsing via paraphrasing."Proceedings of ACL. Vol. 7. No. 1. 2014.
[3] Sagae, K, and Jun'ichi T. "Shift-reduce dependency DAG parsing. Association for Computational Linguistics, 2008.
[4] Berant, Jonathan, et al. "Semantic Parsing on Freebase from Question-Answer Pairs." EMNLP. 2013.

Bachelor Project Supervisor: Mgr. Petr Baudiš

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 16, 2015