Intellexer Question Answering

Aliaksei Bondarionok, Anatoly Bobkov, Liudmila Sudanova, Pavel Mazur, Tatsiana Samuseva
EffectiveSoft Ltd., Minsk, Belarus
contact email: Aliaksei.Bondarionok (a) gmail.com

This is a short description of Intellexer QA system used in QA track 2007. The paper is divided into the following sections that describe modules of the system and certain steps of processing.

1. Intellexer NL parser

The core of Intellexer QA system is a set of modules for natural language processing that perform classic steps of text analysis. The NLP module include: initial preformatting and format conversion (e.g. html to plain text) module, tokenizer, statistical part-of-speech tagger, rule-based POS-tagging corrector, chunker, lexicalized parser, and terms-pairs generator. The system is splitted in two parts: programmatic, written on C++, and linguistic knowledge base (LKB) used by program modules during work. The programmatic part includes certain pre-defined algorithms, and is maintained by software enginners independently from LKB. On the other hand LKB consists of a set of dictionaries, statistically collected data, WordNet, and compiled linguistic rules, and is maintained by computational linguists. There is a set of batch files, perl and shell scripts used to build LKB from source files to binaries suitable for using by the program modules. We provide brief overview of the parser modules below.

1.1) Tokenizer

Tokenizer module is based on a set of context-dependent regular expressions and is able to either insert spaces in text separating words or replace spaces by underscore symbol joining words together into a single token. After this the text is splitted into sentences: a newline character is inserted in certain places, and it works as a sentence separator. So, the text after tokenizer is a set of tokenized sentences – one sentence per line.

1.2) Statistical tagger

We use trigram model [1] for statistical POS-tagging, extended with word statistics (word+tag bigrams and trigrams were used for approximation). A slightly modified Viterbi algorithm is used for finding most probable tag sequence. The model was trained on a large 300M corpus of patents, technical and general texts compiled in a semi-automatic manner. Additionally a manually compiled dictionary of more than 100K wordforms and their possible tags was added to the system to minimize the amount of unknown words. A suffix smoothing algorithm similar to presented in [1] is used to predict tags of unknown words. Our tagset contains 37 POS-tags and is originally derived from LOB (UCREL CLAWS1) tagset [2]. We have merged several LOB POS-tags into a single tag in our tagset (e.g. our tag DT is a combination of LOB tags PN$, WP$, ABL, ABN, ABX, DT, DT$, DTI, DTS, DTX, OD, OD$, PP$, QL, WQL, WDT, WW, WP$R, AT, ATI, AT$). The training corpus consists of only declarative sentences, so the statistical tagger does not perform well on questions. Fortunately, most tagging errors fall into a limited number of classes, that makes it possible to apply rules to correct them. Two examples of POS-tagged sentences are provided below.

The_DT first_DT refracting_NN telescope_NN was_AUX invented_VBN by_IN Hans_NN Lippershey_NN in_IN 1608_CD ._.

Who_WP invented_VB the_DT telescope_NN ?_?

1.3) Rule-based tagging corrector (RBTC)

After the statistical tagger POS-tagged text is processed by RBTC module. The module is based on a set of regular-expression-like rules, that match a certain context and replace found erroneous tag with a correct one. A formalism (regular expressions for tagged text, RETT) for developing such rules was created. For example, a rule for correcting the tag of word "US" from PN (pronoun) to NN (noun) looks like

"(?i:the)"_DT < "(?i:us)"_PN > -> NN.

RBTC module is used both for correcting declarative and interrogative sentences – a separate set of rules is used in each case. The module is able to determine (using a special block of rules) if a sentence is declarative or interrogative and apply the corresponding rule set. In order to be effectively used by program modules RBTC rules are compiled into a combination of finite-state machines (FSM).

1.4) Chunker

The RETT formalism is also used in chunker module. The chunking is performed by a set of cascades [3]. Each cascade is a set of context-dependent rules which is compiled into a binary FSM. Rules identify context and bounds for a noun, verb, adjectival and adverbial phrases in a non-recursive manner and create non-terminal symbols. During chunking a partial parsing tree is generated. For example, a simple rule for forming a noun phrase could be:

< CD* DT|JJ* NN+ > -> wrap NP

In this example, CD, DT, JJ, NN are POS-tags, * - Kleine closure (repeat 0 or more times), + - repeat 1 or more times, < > - brackets that define focus of the rule, wrap NP – a command to form a non-terminal NP from a sequence captured in focus.

In practice, more complex rules are used.

The following examples illustrate chunking.

(NP The_DT first_DT refracting_NN telescope_NN) (VPP was_AUX invented_VBN) by_IN (NP Hans_NN Lippershey_NN) in_IN (DATE 1608_CD) ._.

Who_WP (VP invented_VB) (NP the_DT telescope_NN) ?_?

At the time of submitting results to TREC, chunker did not recognize named entities (NE). During preparations it became clear that NE recognition module should be included into the parser for quality results, but we had strict time frames. Note that chunker recognizes dates.

1.5) Lexicalized parser
After chunking we perform lexicalization of chunks. We extract head word for each chunk and place it together with chunk name. Chunk names perform the same function as a POS-tags in tagged text. POS-tags are also present for words that were not organized into chunks. After lexicalization our examples will look like:

telescope_NP invent_VPP by_IN Lippershey_NP in_IN 1_DATE ._.
Who_WP invented_VP telescope_NP ?_?

A lexicalized chain is then passed to lexicalized parser (lexparser). The module is also based on the RETT formalism. We have developed a comprehensive set of rules for parsing the lexicalized chain, classifying modifiers by type, and building parsing tree. In addition to the regular expression syntax, means for accessing WordNet and statistical PPA resolver plugins were introduced. We believe that accessing information about word's synonyms, hypernyms and hyponyms as well as statistical information help resolve certain types of syntactical ambiguities that cannot be resolved on a syntactical level. WordNet plugin is based on WordNet 2.1 – we have recompiled the file-based semantic graphs into in-memory data structures for faster access. PPA resolver plugin utilizes classic unsupervised statistical approach (using fours VB NN IN NN) for resolving PPA problem [4]. After the lexparser our examples will looks like as follows:

(7_PSVO (telescope_NP The_DT first_DT refracting_NN telescope_NN) (invent_VPP was_AUX invented_VBN) by_IN (Lippershey_NP Hans_NN Lippershey_NN) (001_ADVP-TEMP in_IN (1_DATE 1608_CD))) ._.

(1_QUEST-WHO Who_WP (invented_VP invented_VB) (telescope_NP the_DT telescope_NN)) ?_?

The following parsing trees will be generated (Figs. 1 and 2).

The first refracting telescope was invented by Hans Lippershey in 1608.

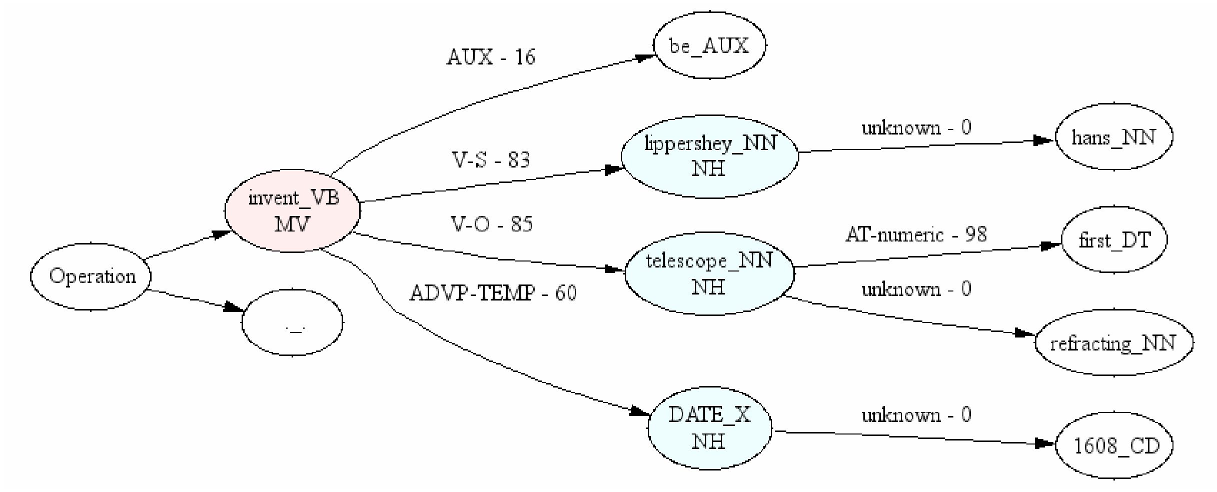

Fig. 1. Parsing tree for a declarative sentence
Legend: Operation – root node, MV – main verb, NH – nominal head, AUX – auxiliary verb tag, VB – verb tag, NN – noun tag, DT – determiner tag, CD – cardinal number tag, X – empty tag, V-S – verb-subject relation, V-O – verb-object relation, ADVP-TEMP – adverbial modifier of time, AT-numeric – numeric modifier, unknown – uncategorized modifier

Who invented the telescope?

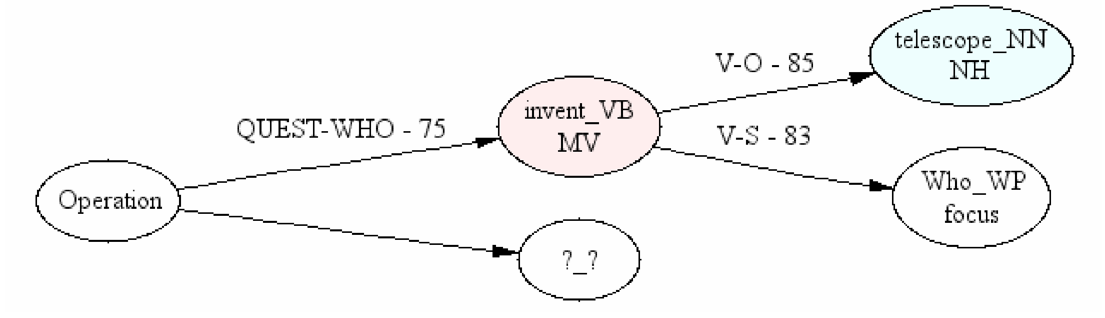

Fig. 2. Parsing tree for a question
Legend: Operation – root node, MV- main verb, NH – nominal head, VB – verb tag, NN – noun tag, WP – wh-pronoun tag, QUEST-WHO – "who" question relation, V-O – verb-object relation, V-S – verb-subject relation, focus – focus of the question

1.6) Paraphraser

It is known that semantically similar ideas may be represented using different syntactic means. For example, "Hans invented the telescope" is the same as "Hans is the inventor of the telescope". According to this idea we have developed paraphrazer module that generates paraphrases of syntactical structures. The module is applied only to user queries, thus increasing recall of question answering at a cost of search time. A formalism that deals with parsing trees was developed, and the module was filled with corresponding rules. Each rule allows to identify a candidate branch in a parsing tree and transforms it into a paraphrased branch (or branches). The paraphrase is then attached to the tree as an alternative for the original branch. We have developed the rules for the most frequent paraphrases. The parsing tree below illustrates paraphrasing (Fig. 3, compare to Fig. 2).

Who invented the telescope?

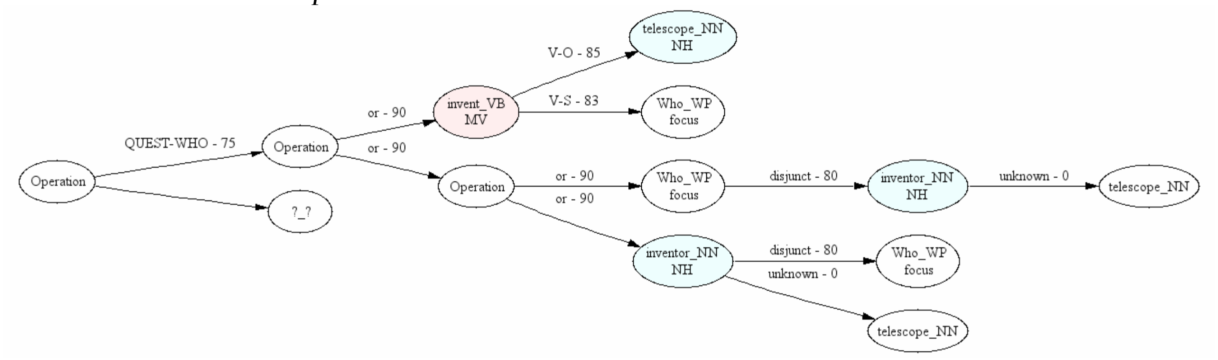

Fig. 3. Paraphrased parsing tree for a question
Legend: Operation – root node, MV- main verb, NH – nominal head, VB – verb tag, NN – noun tag, WP – wh-pronoun tag, QUEST-WHO – "who" question relation, V-O – verb-object relation, V-S – verb-subject relation, disjunct – disjunct ("be")relation, focus – focus of the question

The effect of this paraphrazed tree is the same as instead of asking a question "Who invented the telescope?" we ask three questions (question patterns) simultaneously: "Who invented the telescope?", "Who is the inventor of the telescope?", and "The inventor of the telescope is who?". The table below illustrates questions and corresponding matching answers that have similar tree structure.

Question	**Answer**
Q1. Who invented the telescope?	A1. Hans invented the telescope A2. The telescope was invented by Hans
Q2. Who is the inventor of the telescope?	A3. Hans is the inventor of the telescope
Q3. The inventor of the telescope is who?	A4. The inventor of the telescope is Hans

Before paraphrasing each question matches only answers from the same row (i.e. Q1 matches A1 and A2, Q2 matches A3, Q3 matches A4). After adding paraphrases any given question will match any presented answer (i.e. any of Q1, Q2, Q3 will match all A1, A2, A3, A4).

1.7) Generation of terms and pairs

In order to be effectively indexed and matched parsing trees are disassembled into a set of terms and pairs. Terms are single words taken from nodes of a parsing tree (e.g. *invent*). Pairs are relations between nodes: each pair consists of two words and a dependency between them (e.g. *invent – V-O – telescope*). Pairs for questions and answers are generated in a similar way. For answers we additionally generate half-pairs, where one of the nodes is empty (*e.g. * – V-S – telescope* and *invent – V-S – **). For questions, we generate half-pairs only if one of the nodes is the question focus (e.g. *invent – V-S – **, where * replaces "Who", see Fig. 2 or 3). Half-pairs are used for finding a word in the answer that correspond to the question focus – the word may be retrieved from the corresponding full pair of the answer (*invent – V-S – leppershey*). In plain English this matching process may be illustrated as asking "*Who invented?*" and getting a reply *"leppershey (invented)"*.

For each answer, i.e. sentence from a text collection to be indexed, we generate a list of terms and pairs. For example, below is the full list of terms and pairs generated from the tree at Fig. 1. Terms: *invent, lippershey, hans, telescope, first, refracting, DATE, 1608*. Pairs: *invent – V-S – *, * - V-S – lippershey, invent - V-S – lippershey, lippershey – unknown – *, * – – hans, lippershey – unknown – hans, invent – V-O – *, * – V-O – telescope, invent – V-O – telescope, telescope – AT-numeric – *, * – AT-numeric – first, telescope – AT-numeric – first, telescope – unknown – *, * – AT-numeric – first, telescope – AT-numeric – first, telescope – unknown – *, * – unknown – refracting, telescope – * – refracting, invent – ADVP-TEMP – *, * – ADVP-TEMP – DATE, invent – ADVP-TEMP – DATE, invent – unknown – *, * – unknown – DATE, invent – unknown – DATE, DATE – unknown – *, * – unknown – 1608, DATE – unknown – 1608*.

For each question we generate a list of terms and pairs, and additionally build a tree structure that stores generated pairs (pair tree). Pairs in pair tree are assigned weights according to the type of dependency between its head and modifier nodes (e.g. V-S and V-O dependencies have weight 250). A pair tree may contain empty nodes (with zero weight). Empty nodes perform auxiliary function – show that there are alternative branches (defined by their children). For example, the following pair tree will be generated from the parsing tree at Fig.3 (see Fig. 4).

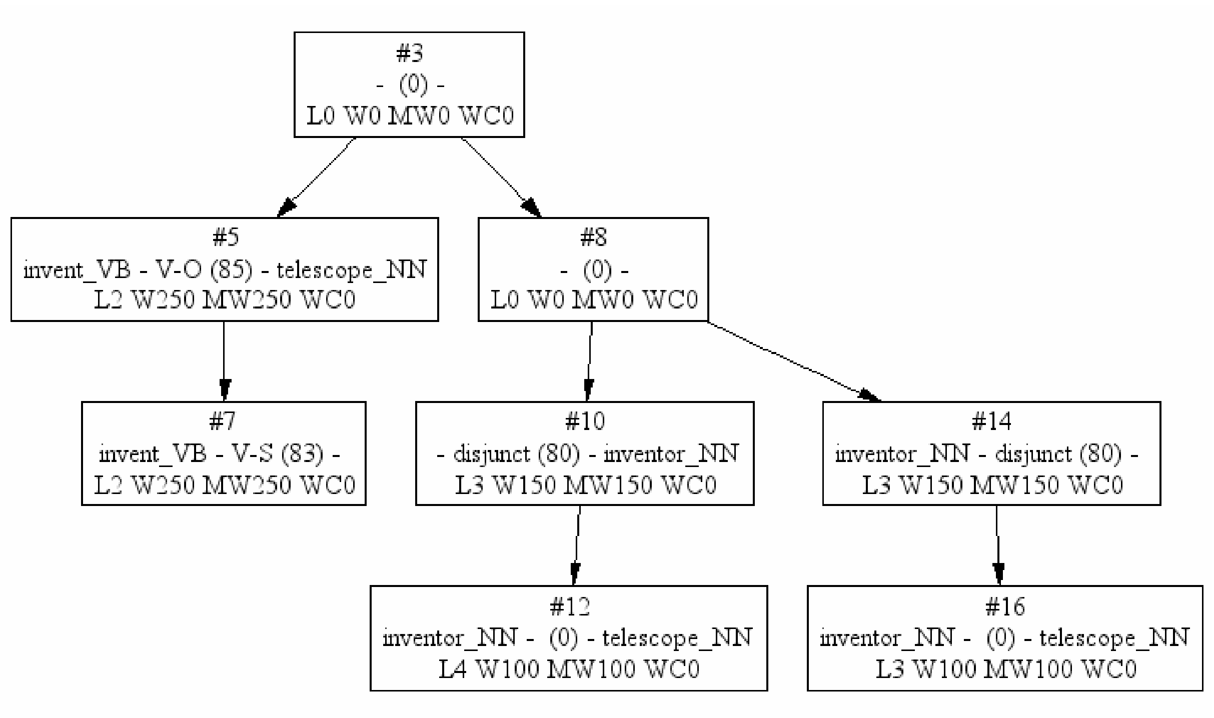

Fig. 4. Pair tree for a question
Legend: L2 – level 2 (from the root), W250 – weight is 250, MW250 – maximum weight among all alternatives of this pair is 250, WC0 – (internal), #3, #5… – (internal) number of node, (85), (0)…- (internal) dependency id

2. Indexing and answering

2.1) Indexing

During indexing stage sentences from an input data collection are transformed into lists of terms and pairs. The lists are then used to build an inverted index “term/pair to sentence id”, where term or pair text is the key and the sentence id is a non-unique value. The index is used during answering for retrieving all sentence ids for a given term or pair. Additionally we create “sentence id to sentence text” and “sentence id to doc id” indexes.

2.2) Resolving anaphora in questions

Before answering we (automatically using a specially designed script) perform anaphora resolution in questions using corresponding topic names. Occurences of pronouns as well as short forms of personal names are replaced with full phrases from the topic. For example, the question “Where did he grow up ?” (question 217.5 from topic “Jay-Z”) will be transformed to “Where did Jay-Z grow up ?”. The question “On what Comedy Central program was Hammond featured ?” (question 218.6 from topic “impressionist Darrell Hammond”) is transformed to “On what Comedy Central program was impressionist Darrell Hammond featured ?”.

2.3) Matching Factoid questions

Answering to factoid questions is performed using the following algorithm.

```
parse a question q and build its term list and pair tree;
let A is a set of answers, A(r,q) - relevance of a result sentence r to the question q;
for each path p (from the root to a leaf) in the pair tree do:
    let P(r,p) – relevance of result r to the path p;
    calculate maximum path weight M(p), which is the sum of MW values of pairs on the path p;
    for each pair e of the path p do
        retrieve corresponding sentence ids {$r_i$} using “term/pair to sentence id” index
        for each retrieved sentence id $r_i$ do:
            P{$r_i$,p} += W(e), where W is the weight of current pair e in question pair tree;
    normalize values of P{$r_i$} by dividing them by M(p);
    append values from P to A (ignoring low-relevance values)
optionally perform (1) and (2) – see below
select the most relevant answer r from A, if A(r,q) below certain threshold, the answer will be NIL
if answer is not NIL
    retrieve sentence text using “sentence id to sentence text” index;
    process sentence text, build parsing tree and generate term and pair list for it;
    find the word in the answer that matches the focus of the question as mentioned in 1.7;
    for the focus word, get its attribute nodes in the parsing tree of the answer (e.g. if the focus word is
    “lippershey”, its attribute will be “hans”, see Fig. 1), and return the focus word and its attributes as
    the direct answer to the question.
```

(1) After matching by pairs as described above we perform search using terms in a similar manner. The only difference is that terms for the question are organized in plain list instead of tree and therefore we have only one alternative. After matching by terms the results are combined with those received by pairs. Results received by terms are considered 7 times less imprortant that received by pairs, i.e. the total proximity of a result r to a question q will be a weighted average $A(r,q) = 7/8 * A_{pairs}(r,q) + 1/8 * A_{terms}(r,q)$. Matching by terms is not directly related to answering a question, but increases recall of the system when we have few high-relevant results by providing more approximate results.

(2) In addition to the described algorithm for TREC QA track we perform topic trimming. First, the text of a topic is parsed as a declarative query, and a set of terms and pairs is generated. Then terms and pairs are searched in the inverted index “term/pair to sentence id” and we calculate T(d,t) proximity of a document d to topic t. After that we start processing queries from the given topic, and each A(r,q) is multiplied by T(d(r),t), where d(r) is the document containing sentence r. So the final proximity of result to query will be $A_{final}(r,q) = A(r,q) * T(d(r),t)$. In other words this approach filters out documents that are not related to the given topic, and re-estimates proximity of the results from the remaining documents.

2.4) Matching List questions

Matching list questions is performed in the same manner as matching factoids. The difference is that we save not a single answer, but a list of answers with relevances above a certain threshold. Since we cannot return NIL response for list questions, in the case if we do not have any direct answers (i.e. words or phrases that directly correspond to question focus) returned we apply heuristic. The heuristic scans answer sentences for possible matches. The question is lexically seached for certain patterns, and depending on this the heuristic tries to extract certain kind of entries from answer sentences. For example, if the question starts with “when” the heuristic will look for dates, “what year” – 4-digit words starting from 1 (1\d\d\d), “who” – capitalized sequencies where first word is on the list of popular personal names, “how many” – numbers, etc. Then all found entries are sorted and counted. After that we select at most 5 most frequent of them that do not intersect the topic name and return as list response.

2.5) Matching Other questions

Other questions are answered using topic names as queries. A topic name is parsed, a pair tree is built and search is performed. The returned set of result sentences is trimmed to contain only sentences that did not previously appear in the corresponding factoid and list questions from the topic. After that we generate snippets from each answer sentence using the following steps:

1. Build a parsing tree for the sentence (e.g. see Fig. 1).
2. Locate words in the sentence parsing tree that match words from the topic pair tree (e.g. if the topic is “hans lippershey” we will locate two nodes “hans” and “lippershey”, see Fig. 1)
3. Retrieve surrounding V-S (subject-verb) and V-O (verb-object) relations for the matched words (e.g. “invent” and “telescope” will be found)
4. Extract snippets that consist of matched words and correponding V-S/V-O relations from the sentence plain text (e.g. “telescope was invented by Hans Lippershey” will be extracted).

Spippets from all sentences are combines together, sorted and counted. Then we select several (up to 10) most frequent snippets that are about 5-6 words in length and return them as a response to the Other question.

2.6) Difference between runs A, B and C

The scheme described above is used to generate Intellexer7A run. Runs B and C have the following minor differences:

Run B: when choosing the best answer for a factoid question, the preference is given to answers represented by a) sequences of two or more uppercased words; b) single uppercased words; and then to c) other sequences.

Run C is the same as run A, but uses slightly different weighting scheme for generation of snippets for other queries.

[1] Thorsten Brants, 2000. TnT - A Statistical Part-of-Speech Tagger. In Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000, Seattle, WA.
[2] UCREL CLAWS1 (LOB) Tagset - http://www.comp.lancs.ac.uk/ucrel/claws1tags.html
[3] Steven Abney. Partial Parsing via Finite-State Cascades. In Proceedings of the ESSLLI '96 Robust Parsing Workshop, 1996.
[4] Adwait Ratnaparkhi. Statistical Models for Unsupervised Prepositional Phrase Attachment. http://acl.ldc.upenn.edu/P/P98/P98-2177.pdf