# Mining Knowledge from Large Corpora for Type Coercion in Question Answering

James Fan, Aditya Kalyanpur, J. William Murdock and Branimir K. Boguraev

IBM Research, 19 Skyline Drive, Hawthorne NY 10532
{fanj, adityakal, murdockj, bran}@us.ibm.com

**Abstract.** A fundamental issue in natural language processing and knowledge acquisition is recognizing whether a given string is an instance of a given type. In this paper, we present a solution to the typing problem by mining knowledge from an unstructured text corpus, and apply it in the context of type coercion in question answering. We describe a new generate-and-type framework, called TyCor (short for Type Coercion), in which candidate answers are initially produced without the use of answer type information, and subsequent stages check whether the candidate answer's type can be coerced into the Lexical Answer Type (LAT) of the question. The TyCor framework consists of a suite of type checking algorithms and we describe two specific implementations that leverage the regularity and redundancy of linguistically expressed type relationships in a large corpus. We evaluate the performance of these TyCor algorithms for entity-typing over a broad domain test set and demonstrate their impact on end-to-end performance of our DeepQA system on blind Jeopardy! data.

**Keywords:** Question Answering, Type Checking, Open IE, Ontologies

## 1 Introduction

Typing, the task of recognizing whether a given string is an instance of a given type is a fundamental problem in artificial intelligence (AI). Solving this problem in an open domain has implications for many areas in AI, such as reasoning, natural language processing, and knowledge acquisition. In this paper, we present a solution to the typing problem by mining knowledge from an unstructured text corpus, and apply it in the context of type coercion in question answering.

Many open domain question answering (QA) systems have relied on a notion of *Predictive Annotation* [1] in which a fixed set of expected answer types are identified through manual analysis of a domain, and a background corpus is automatically annotated with possible mentions of these types before answering questions. These systems then analyze incoming questions for the expected *answer type*, mapping it into the fixed set used to annotate the corpus, and restrict candidate answers retrieved from the corpus to those that match this answer type using semantic search (IR search augmented with the ability to search for words tagged with some type).

A large number of QA systems that have been developed under the influence of organized standard evaluation efforts such as TREC/TAC[1], CLEF[2], and NTCIR[3] adopt this type-and-generate approach, which suffers from several problems. First, restricting the answer types to a fixed and typically small set of concepts makes the QA system brittle and narrow in its applicability and scope. Such a closed-typing approach does not work for open-domain Question Answering, and in particular the Jeopardy! problem, where answer types in questions span a very broad range of topics, are expressed using a variety of lexical expressions (e.g. "scarefest" when referring to horror movies) and are sometimes vague (e.g. "form") or meaningless (e.g. "it"). Second, the QA system performance is highly dependent on the precision and recall of the predictive annotation software used, which acts as a candidate selection filter. In addition, when questions ask for types of answers not covered by the fixed set of types, the QA system either fails to generate answers at all, or uses some catchall type (e.g. "OTHER") for which the rest of the system is typically not well suited. In other words, performance on questions whose answer types are outside the fixed type system is significantly worse than when the answer type is in the type system.

In contrast to the type-and-generate approach, we consider a *generate-and-type* framework, in which candidate answers are initially produced without use of answer type information, and subsequent stages check whether the candidate answer's type can be *coerced* into the Lexical Answer Type (LAT) of the question. The framework is based loosely on the notion of *Type Coercion* [2] (TyCor). The most notable aspects of the approach are: it does not rely on a fixed type system, however it does not discard one when available and useful; it is a multi-strategy and multi-source approach, gathering and evaluating evidence in a generative way rather than a predictive one; it is not part of candidate generation, rather it is simply another way of analyzing and scoring candidate answers; it is not a hard filter, producing instead for each candidate answer a probability that it is (or is not) of the right type.

When not relying on a fixed type system, developing a system that has good instance coverage for a wide variety of types becomes a key design point. In this paper, we present a novel typing framework, called TyCor. We provide a high-level architecture of the TyCor framework and discuss how it is integrated in our open-domain QA system, known as DeepQA [3]. The TyCor framework consists of a suite of type checking algorithms and we describe two specific implementations that leverage the regularity and redundancy of linguistic information in large corpora for typing. Since the corpora we use are large in size, we can obtain type information for a large set of instances, and the resulting TyCor components can provide coverage for even rare types.

We briefly describe the TyCor algorithms for the two components, evaluate their performance for entity-typing over a broad domain test set, and demonstrate their impact on end-to-end performance of our DeepQA system on blind Jeopardy! data.

---

[1] http://trec.nist.gov/

[2] http://www.clef-campaign.org/

[3] http://research.nii.ac.jp/ntcir/index-en.html

The key contribution of this paper is the definition of a novel Type Coercion (TyCor) framework for open domain Question Answering, and showing how lexical type information derived from a large text corpus (using traditional techniques) can be integrated into this TyCor framework to boost end-to-end QA performance.

## 2 Background: Open Domain Question Answering

Our aim was to build a QA system capable of rivaling expert human performance at answering open-domain questions. As a test bed, we chose the challenging TV quiz show Jeopardy!, whose questions cover a wide range of topics and are expressed using rich, complex natural language expressions. A characteristic of Jeopardy! questions is that nearly any word in the English language can be used as an answer type, e.g. consider:

- Invented in the 1500s to speed up the game, this *maneuver* involves 2 pieces of the same color. (answer: castling)
- The first known airmail service took place in Paris in 1870 by this *conveyance*. (answer: hot-air balloon)
- When hit by electrons, a phosphor gives off electromagnetic energy in this *form* (answer: light)

Given this variability, it is to be expected with predictive annotation that we cannot reliably predict what types there are going to be and what their instances are. We need to be open and flexible about types, treating them as a property of the question and answer combined. In other words, instead of finding candidates of the right type, we want to find candidates (in some way) and judge whether each one is of the right type by examining it in context with the answer type from the question. Furthermore, we need to accommodate as many sources as possible that reflect the same descriptive diversity as these questions. These were the underlying principles of the type coercion methodology in our QA system, known as DeepQA.

### 2.1 DeepQA

DeepQA is a massively parallel probabilistic evidence-based architecture designed to answer open domain questions. It consists of the following major stages (more details can be found in [3]):

**Question Analysis:** The first stage of the pipeline performs a detailed analysis to identify key characteristics of the question (such as focus, lexical answer type, question class, etc.) used by later stages of the pipeline. The focus is the part of the question that refers to the answer, and typically encompasses the string representing the lexical answer type (LAT). The system employs various lexico-syntactic rules for focus and LAT detection, and also uses a statistical machine-learning model to refine

the LAT(s). Like all parts of our system, LAT detection includes a confidence, and all type scores are combined with LAT confidence

**Hypothesis (Candidate) Generation:** One of the significant differences between our approach and that of many previous QA systems is the manner in which candidate answers are generated. In type-and-generate based approaches, the question is analyzed and a semantic answer type (SAT) from a predefined set of types is identified. A background corpus, which has been pre-annotated with these types, is searched using keywords from the question and the SAT as a required term. In other words, only candidate answers that, during corpus analysis, are known to match the SAT, will be returned as candidates.

In our QA system, processing of type information has been moved from candidate generation to answer scoring. For the candidate generation step, the system issues several queries derived from question analysis results to search its corpus for relevant documents and passages, and it uses a variety of candidate generators to produce a list of answer candidates to the question.

**Hypothesis and Evidence Scoring:** Answer scoring is the step in which all candidates, regardless of how they were generated, are evaluated. During the answer-scoring phase, many different algorithms and sources are used to collect and score evidence for each candidate answer. Type information is just one kind of evidence that is used for scoring. Other dimensions of evidence include temporal and spatial constraints, n-grams, popularity, source reliability, skip-bigrams, substitutability, etc.

**Candidate Ranking:** Finally, machine-learning models are used to weigh the analyzed evidence and rank the answer candidates. The models generate a confidence for each answer candidate being the correct answer to the given question, and the system answers with the top-ranked candidate. The system can also choose to refrain from answering if it has a low confidence in all of its candidates.

### 2.2 Type Coercion (TyCor)

The TyCor framework consists of a suite of answer scoring components that take a Lexical Answer Type (LAT) and a candidate answer, and return a probability that the candidate's type is the LAT, using a collection of unstructured, semi-structured and structured sources. Note that since language does not distinguish between instantiation and subclass, the TyCor components must allow for this, and given a candidate answer that refers to a class, it should be given a high score if it can be interpreted as a subclass or instance of the LAT. As mentioned in the previous subsection, LAT detection produces a confidence, so each (answer, LAT) score is modified by the LAT confidence.

Each TyCor component uses a source of typing information and performs four steps:

*Entity Disambiguation and Matching (EDM):* The most obvious, and most error-prone, step in using an existing source of typing information is to find the entity in that source that corresponds to the candidate answer. This step is especially critical for TyCor components that rely on structured or semi-structured resources. Since the candidate is just a string, this step must account for both polysemy (the same name may refer to many entities) and synonymy (the same entity may have multiple names). Each source may require its own special EDM implementations that exploit properties of the source, for example DBPedia encodes useful naming information in the entity id. EDM implementations typically try to use some context for the answer, but in purely structured sources this context may be difficult to exploit.

*Predicate Disambiguation and Matching (PDM):* Similar to EDM, this step finds the type in the source that corresponds to the LAT found. In some sources this is the same algorithm as EDM; in others, type looking requires special treatment. In a few, especially those using unstructured information as a source, the PDM step just returns the LAT itself. In type-and-generate, this step corresponds to producing a semantic answer type (SAT) from the question. PDM corresponds strongly to notions of word sense disambiguation with respect to a specific source.

*Type Retrieval (TR):* Once an entity is retrieved from the source, if applicable the types of that entity must be retrieved. For some TyCors, like those using structured sources, this step exercises the primary function of the source and is simple. In others, like unstructured sources, this may require parsing or other semantic processing of some small snippet of natural language.

*Type Alignment:* The results of the PDM and TR steps must then be compared to determine the degree of match. In sources containing e.g. a type taxonomy, this may include checking the taxonomy for subsumption, disjointness, etc. For other sources, alignment may utilize resources like Wordnet for finding synonyms, hypernyms, etc. between the types.

Each of the steps above generates a score reflecting the accuracy of its operation, taking into account the uncertainty of the entity mapping or information retrieval process, and the final score produced by each TyCor component is a combination of the four step scores. This score is typically computed as the product, though a more effective combination can be learnt using machine learning by training on corresponding ground truth (entity typing data).

## 3 Mining Knowledge from Unstructured Sources

The regularity and redundancy of linguistically expressed is-a information in a large amount of unstructured sources often provide direct evidence that an answer is a member of a LAT, and thus can be very informative for the type coercion task. For this reason, we built a large shallow semantic resource called Prismatic that was

automatically extracted from text, specifically leveraging an isa relation detector using syntactic patterns which was applied as part of the text processing.

### 3.1 Prismatic Resource

Prismatic [4] is a large scale lexicalized relation resource mined from over 30 GB of text. It is built using a suite of NLP tools that includes a dependency parser, a rule based named entity recognizer and a coreference resolution component. Prismatic is composed of frames which are the basic semantic representation of lexicalized relations and their syntactic context. There are approximately 1 billion frames in our current version of Prismatic.

Prismatic is built using a three step process: corpus processing, frame extraction and frame cut extraction. Figure 1 outlines the Prismatic pipeline.

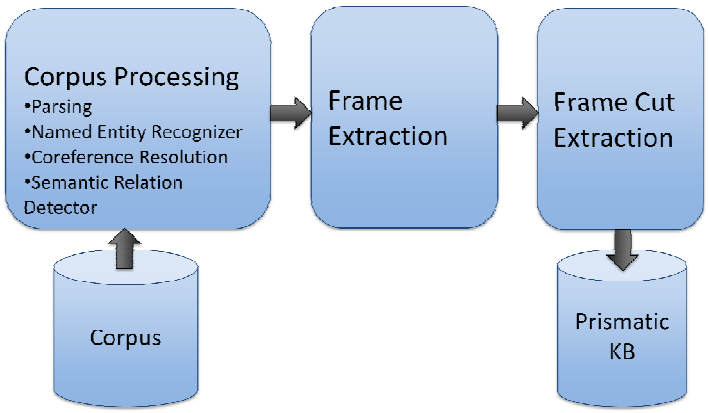


**Figure 1. The Prismatic pipeline**

The key step in the Corpus Processing stage is the application of a dependency parser which is used to identify the frame slots for the Frame Extraction stage. A rule based Named Entity Recognizer (NER) is used to identify the types of arguments in all frame slot values. This type information is then registered in the Frame Extraction stage to construct intentional frames. Additionally, semantic relations, such as 'isa' relations in particular, are extracted by semantic relation detectors.

The next step of Prismatic is to extract a set of frames from the parsed corpus. A frame is the basic semantic unit representing a set of entities and their relations in a text snippet. A frame is made of a set of slot value pairs where the slots are dependency relations or semantic relations extracted from the corpus processing step and the values are the terms from the sentences or types they are annotated with. Table 1 shows the extracted frame based on the parse tree in figure 2.

One of the main reasons for extracting a large amount of frame data from a corpus is to induce interesting knowledge patterns by exploiting redundancy in the data. For example, we would like to learn that things that are annexed are typically regions, i.e., a predominant object-type for the noun + preposition "annexation of" is "Region" where "Region" is annotated by a NER. To do this kind of knowledge induction, we first need to abstract out specific portions of the frame—in this particular case, we need to isolate and analyze the noun + preposition + object-type relationship. Then, given a lot of data, and frames containing only the above relationship, we hope to see the abstracted frame [noun="annexation", preposition="of", object-type="Region"] occur very frequently.

To enable this induction analysis, we define frame-cuts. For example, we define an N-P-OT frame cut, which when applied to a frame only keeps the noun (N), preposition (P) and object-type (OT) slots, and discards the rest. Similarly, we define frame-cuts such as S-V-O, S-V-O-IO, S-V-P-O etc. (where S - subject, V - verb, O - object, IO - indirect object). For example, we can use the V-OT frame cut to learn that a predominant object-type for the verb "annex" is also "Region", by seeing lots of frames of the form [verb="annex", object-type="Region"] in our data. Frames can specify optional value constraints for slots. For example, we can define an S-V-O frame cut, where both the subject (S) and object (O) slot values are constrained to be proper nouns, thereby creating strictly extensional frames, i.e. frames containing data about instances, e.g., [subject="United States" verb="annex" object="Texas"]. The opposite effect is achieved by constraining S and O slot values to common nouns, creating intensional frames such as [subject="Country" verb="annex" object="Island"]. The separation of extensional from intensional frame information is desirable, both from a knowledge understanding and an applications perspective, e.g. the former can be used to provide factual evidence in tasks such as question answering, while the latter can be used to learn entailment rules as seen in the "annexation" case.

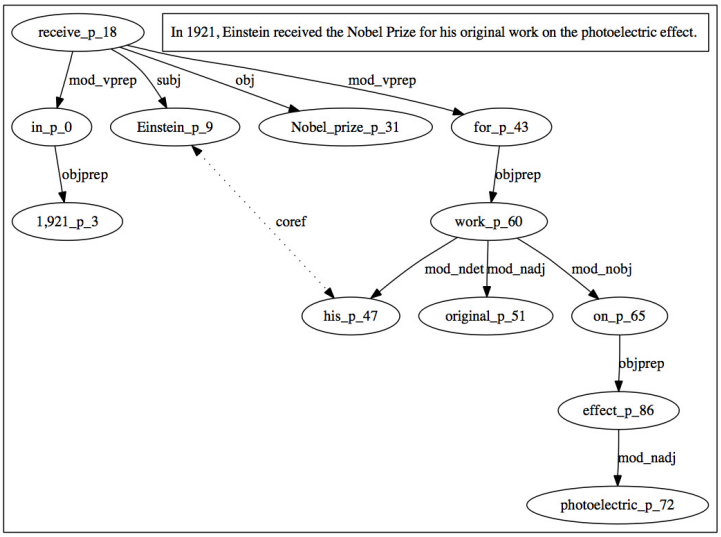


**Figure 2. The parse tree of the sentence *In 1921, Einstein received the Nobel Prize for his original work on the photoelectric effect.***

| **Frame 01** | |
|---|---|
| verb | receive |
| subj | Einstein |
| type | PERSON / SCIENTIST |
| obj | Nobel Prize |
| mod_vprep | In |
| objprep | 1921 |
| type | YEAR |
| mod_vprep | For |
| objprep | Work |

| **Frame 02** | |
|---|---|
| noun | Work |
| mod_ndet | His / Einstein |
| mod_nobj | On |
| objprep | Effect |

**Table 1. Frames extracted from Dependency Parse in Figure**

### 3.2 Syntactic pattern based IS-A relation detection

A semantic relation can be conceptualized as a labeled edge between two terms. Processes like search query generation, answer lookup, passage scoring, semantic constraints inference, and others, benefit from such labeled associations between text elements, which abstract away from the syntax and normalize over the numerous ways in which essentially the same facts can be expressed with language [5], [6], [7].

We distinguish between deep and shallow semantic relations. Deep semantic relations appeal to a specific domain (like relationships between animate agents and works of art in an arts-and-entertainment domain, [8, 9]). Shallow semantic relations do not so much map to a particular type ontology, but instead detect certain syntactic contexts indicative of alternative surface realizations for essentially the same semantic relationship ([6]). Within the latter category is a set of universal ontological relations, such as 'instanceOf', 'subtypeOf', 'sameAs', which are all interpreted as variations within the 'isa' detection task for Prismatic. Examples of these are found in questions like:

- Giuseppina Streppoini, a prima donna in "Nabucco", married this famous Guiseppi in 1859.
- Very simply, it's any plant, such as crabgrass, that grows where it's not wanted.
- John F. Kennedy was elected the first Catholic president of the United States.

Our relational analysis would derive, respectively,

(1) instanceOf(prima donna, Giuseppina Streppoini),
(2) subtypeOf(plant, crabgrass),
(3) sameAs(John F. Kennedy, president).

At the level of recognition, we maintain some distinction between the different relations. Thus, one of the arguments of 'instanceOf' will typically be a named entity, while both arguments of 'subtypeOf' will be lexical types. The 'sameAs' relation captures a broader range of coreferencial locutions, mediated by lexical context of verbs like "be", "become", "elect", and so forth. Such distinctions are important to some DeepQA components; as far as Prismatic is concerned, however, all three extractions above are instances of an 'isa' relation.

A relation can often be expressed in multiple ways lexically and syntactically. In many cases, however, basic principles of economy of expression and/or conventions of genre will ensure that certain systematic ways of expressing a relation are repeatedly used, to the extent that regularities in expression can be observed and categorized. This has a particular consequence in the knowledge mining setting for Prismatic, where the sheer volume and redundancy of data will ensure that even with

a relatively small number of high-precision patterns, a large number of relation instances will be detected, to reliably inform the aggregation processes.

We have, therefore, chosen to build a rule-based 'isa' recognizer (building upon the three relation types illustrated above), based on a relatively small number of precise—and productive—patterns, in the tradition of [10]. We are aided in this by our state-of-the-art parser [8], which can correctly identify appropriate syntactic configurations and pinpoint pattern elements as eventual arguments.

We have identified five main recognition rule groups (shown below) to capture the "isa" relation. Each rule group targets a set of distinct syntactic realizations of relation expressions characterizing the group. Below we provide illustrative examples for each of the groups.

appositiveRule :

- The prophet of this religion, Mirza Husayn Ali, is known as Baha'u'llah or "Glory of God" to the faithful.
  - `instanceOf(prophet, Mirza Husayn Ali).`

includesRule :

- This author's thrillers turned into films include "Phantoms" and "Demon Seed".
  - `instanceOf(thriller, Phantoms),`
  - `instanceOf(thriller, Demon Seed).`
- Some woody plants like conifers reproduce by means of exposed seeds.
  - `subtypeOf(plant, conifer).`

catInstanceRule :

- Singer/actor Sting played Feyd-Rautha in this David Lynch film based on a Frank Herbert novel.
  - `instanceOf(singer, Sting),`
  - `instanceOf(actor, Sting).`
- Young filmmakers Daniel Myrick and Eduardo Sanchez created quite a frenzy with this film.
  - `instanceOf(filmmaker, Daniel Myrick),`
  - `instanceOf(filmmaker, Eduardo Sanchez).`

sameAsRule :

- Deng Xiaoping becomes the first top Communist leader from China to visit the US.
  - `instanceOf(leader, Deng Xiaoping).`

altNamesRule :

- This itchy inflammation is medically known as conjunctivitis.
  - `subtypeOf(iflammation, conjunctivitis).`
- The American Indian name for this peak is Denali.
  - `instanceOf(peak, Denali).`

## 4 Prismatic TyCor

In order to utilize the knowledge in a large corpus for the type coercion task, we augmented the Prismatic resource to include 'isa' relations using the syntactic pattern based relation detector as part of the corpus processing step. Consequently, the frames extracted from the corpus can include 'isa' relations as well. The 'isa' relation is usually annotated between two nouns to indicate that the first noun is of the type denoted by the second noun, and is captured using a noun-isa frame cut.

We have developed a TyCor component to use the augmented Prismatic. It takes a candidate answer and a lexical answer type (LAT) as input, and returns a score indicating the likelihood of the candidate answer being an instance of the LAT. The Prismatic TyCor computes this score *p* as the following:

$$p = \frac{C_{matched}}{C_{total}}$$

where $C_{matched}$ is the number of 'isa' relation occurrences that show the candidate answer is of the LAT in Prismatic, and $C_{total}$ is the total number of 'isa' relation occurrences that show the candidate answer is of any type in Prismatic.

Unlike TyCor components that use structured or semi-structured resources, the Entity Disambiguation and Matching step, the Predicate Disambiguation and Matching step and Type Retrieval step are very simple. Prismatic TyCor does not use any type system, and it uses the candidate answer string as the candidate type after the Type Retrieval step. Similarly, it does not map the LAT to a type in a type system; it uses the LAT string as the type after the Predicate Disambiguation and Matching step. Prismatic TyCor does the bulk of its work during the Type Alignment step.

Instead of relying on predefined taxonomy for subsumption tests, Prismatic TyCor utilizes the subsumption relations directly expressed between two strings in large amount of text. This approach has advantages and disadvantages. First, the large corpora size allows Prismatic TyCor to have wider coverage than most fixed taxonomies. This is particular useful in open domain QA, such as Jeopardy! because of the many rare LATs. Second, its results are not compounded by the errors in EDM, PDM or TR steps since these steps are trivial in Prismatic TyCor. The main source of errors is from the 'isa' relation detector which is mitigated by the redundancy in large corpus. This approach also has a disadvantage—it does not take the context of the question into consideration. Prismatic TyCor uses the overall aggregate statistics from the whole corpus, but it does not consider any clue contexts. For example, although the string *George Washington* may refer to a president, a general, a bridge, etc., in a particular question involving college basketball, *George Washington* is more likely to be a university basketball team. Another TyCor component, Passage TyCor, addresses

this potential deficiency by using both context and the knowledge mined from unstructured text, and it is described next.

## 5 Passage TyCor

Passage TyCor uses the same syntactic pattern-based relation detector that Prismatic TyCor uses. However, unlike Prismatic TyCor, Passage TyCor only considers occurrences of these syntactic patterns in contexts that DeepQA's passage search algorithms [6] have identified as relevant to the question being asked. This approach provides less coverage than Prismatic TyCor, since it is using only a small fraction of our full text corpus. However, it has a significant advantage when dealing with ambiguous answers or types, because the senses of the words used in the passages found by search are likely to be the ones that are most relevant to the question being asked.

In addition, we configure Passage TyCor to be more aggressive in its Type Alignment than we do for Prismatic TyCor. Specifically, we use a variety of resources to determine equivalent lexical types. This partially compensates for the fact that we use only a few selected passages from the corpus as our source of type information. For example, if we have a question asking for a *building* and a retrieved passage saying "Rockefeller Center is one of the most spectacular edifices in midtown Manhattan," we will conclude that "Rockefeller Center" is an answer that satisfies the question's type requirement. Passage TyCor draws this conclusion because the syntactic pattern-based relation detector (run during preprocessing) finds an 'isa' relation between "Rockefeller Center" and "edifice" in this passage, and then at run time (after DeepQA's passage retrieval has determined that this passage appears to be relevant to the clue), Passage TyCor uses its synonym resources to determine that "building" and "edifice" are synonymous.

## 6 Experiments

### 6.1 Impact on End-to-End Question Answering

Table 2 shows the impact of the Prismatic and Passage TyCor answer scorers on end-to-end Jeopardy! question answering. The data shows accuracy (percentage of all questions that are answered correctly) on 3,508 randomly selected, previously unseen Jeopardy! questions. The first set of results are from a baseline system that includes all of DeepQA's question analysis, search, and candidate generation, but none of its answer scoring components except for the ones listed in the column headings. The second set of results are from the full Watson question answering system with all of the answer scoring components except for TyCor components. In both sets, the left column is for a system that includes no TyCor component, the second column

includes only Prismatic TyCor, the third includes only Passage TyCor, and the fourth includes Prismatic and Passage TyCor.

| | No TyCor | Prismatic TyCor | Passage TyCor | Both TyCors |
|---|---|---|---|---|
| Baseline System | 50.1% | 53.9% (+3.9%) | 51.2% (+1.1%) | 54.6% (+4.5%) |
| Full System | 65.6% | 68.0% (+2.4%) | 66.6% (+1.0%) | 68.1% (+2.5%) |

**Table 2 Impact of Prismatic and Passage TyCor answer scorers on question answering accuracies**

In all cases, the difference between either/both TyCor and no TyCor is statistically significant ($p < .05$ using McNemar's test with Yates' correction for continuity). The impact of both TyCors is significantly greater than the impact of either TyCor alone in the baseline configuration (only). The baseline configuration is generally better for measuring subtle distinctions such as showing (as we do here) that Passage TyCor and Prismatic TyCor are complementary in the sense that an end-to-end question answering system can perform better with both components than it does with either component alone. The results on the full system are more muted but are useful for getting a more realistic sense of the rough magnitude of the usefulness of these components in a complete system.

## 8. Related Work

TextRunner [11] is an information extraction system which automatically extracts relation tuples over a massive web dataset in an unsupervised manner. TextRunner contains over 800 million extractions [12] and has proven to be a useful resource in a number of important tasks in machine reading such as hypernym discovery [13], and scoring interesting assertions [12]. TextRunner works by automatically identifying and extracting relationships using a conditional random field (CRF) model over natural language text. As this is a relatively inexpensive technique, it allows rapid application to web-scale data.

DIRT (Discovering Inference Rules from Text) [14] automatically identifies inference rules over dependency paths which tend to link the identical arguments. The technique consists of applying a dependency parser over 1 GB of text, collecting the paths between arguments and then calculating a path similarity between paths. DIRT has been used extensively in recognizing textual entailment (RTE).

Never Ending Language Learner (NELL) [15] is another system that acquires knowledge automatically. It starts with a set of seed categories and relations with a handful of examples, and it extracts new instances of categories and relations from millions of web pages.

Another recent system, Background Knowledge Base (BKB) [16], also extracts parts of the parsed structure from a domain specific corpus as knowledge source to improve machine reading systems.

Prismatic is similar to TextRunner and DIRT in that it may be applied automatically over massive corpora. At a representational level it differs from both

TextRunner and DIRT by storing full frames from which n-ary relations may be indexed and queried. Prismatic differs from TextRunner as it applies a full dependency parser in order to identify dependency relationships between terms. In contrast to DIRT and TextRunner, Prismatic also performs co-reference resolution in order to increase coverage for sparsely-occurring entities and employs a named entity detector (NED) and relation extractor on all of its extractions to better represent intensional information.

Prismatic is similar to NELL in the sense that it also extracts instances of categories and relations automatically from a large corpus. However, the categories and relations are open-ended for Prismatic; they are not limited to the set of given examples for NELL.

Prismatic is similar to BKB in that both use parser outputs to extract knowledge. They differ in terms of the corpora they use and their applications. BKB uses a domain-specific corpus to focus on aggregate statistics relevant to that domain to improve machine reading systems' performance. Prismatic uses an open domain general purpose corpus to obtain overall aggregate statistics of text usage.

Automatic detection of hyponymy (isa) and meronymy (part-of) relations mark the oldest work in relation extraction. The idea of exploting the regular patterns specifying a relationship between two entities originally belongs to Amsler [17], who leveraged the structure of dictionary definitions for automatic taxonomy construction. This was implemented, on a large scale, by Chodorow et al. [18], as a system for automatic computational lexicon construction. These were systems with virtually 100% precision (every entry in a dictionary is correct), but less than desired coverage (a dictionary only has so many entries). With the emergence and growth of on-line text, Hearst [10] pioneered the natural extension of the idea, focusing on patterns in freely occurring text.

This idea has had several different incarnations, all motivated by the increasing volume of on-line resources: a situation where noise mandates seeking a balance between high precision and high recall of reliably identifying relation instances, preferably with minimal supervision. The overall trend has been to derive patterns from seeds: for instance, Ravichandran and Hovy [19] use the notion of pairing question elements with their answers, across a range of common question types. Scaling this strategy to the Web yields reasonable relation instances for specific relations like 'birthdate', 'inventorOf', and 'location'; noise, however, leads to low precision--especially for not so specific relations, like 'isa'. More recently, Pantel and Pennachiotti [20] propose a different notion of seeding, grounded in "generic patterns": i.e. patterns with high recall and low precision. The noise which comes with broad coverage is counteracted by a novel filtering algorithm; again, the scale of the Web facilitates noise control. The work seeks applicability to a set of relations broader than just 'isa'.

Our approach, already different in the special prominence it gives to finding instances of 'isa' in particular, also appeals to the scale of the Web, remaining closest to Hearst's original proposal. We have a relatively small number of (precise) patterns, but assume that the redundancy of on-line data will generate more than enough

instances whose aggregate statistics will support probabilities of them being correct. Also, our 'isa' relation detection is done by analyzing a deep syntactic parse of the sentence (in particular, the predicate argument structure [8]), whereas earlier work (starting with Hearst's patterns) is based on shallower matching strategies, such as constituent analysis over part-of-speech tags stream, and/or other linear pattern interpretation techniques. Finally, in contrast to most of the work seeking to identify 'isa' relations, ours is embedded in a component of an operational open domain question answering system.

## 9. Conclusion and Future Work

In this paper, we have presented a broad domain solution to the entity typing problem by mining knowledge in a large unstructured text corpus, and applied it in the context of type coercion in question answering. We introduced the generate-and-type framework used in the DeepQA question answering system. This framework allows candidate answers to be produced without the use of answer type information, with subsequent stages (TyCor components) determining whether the candidate answer's type can be coerced into the Lexical Answer Type of the question. We then described how lexical type knowledge from a large background knowledge source (Prismatic) and from supporting passages for candidate answers can be integrated into the TyCor framework. Our evaluation on 3,508 randomly selected, previously unseen Jeopardy! questions shows that our corpus based TyCor solution has significant end-to-end impact on QA performance.

In the future, we plan to augment the frames in Prismatic with contextual information which shall enable Prismatic TyCor to take the question context into consideration during matching. We also plan to include a statistical relation detector in addition to the pattern based relation detector to increase the coverage of "isa" relation annotation. We expect these two changes to increase both the precision and recall of the Prismatic TyCor.

## References


[1] Prager, J.M., Brown, E.W., Coden, A. and Radev, R. (2000) Question-Answering by Predictive Annotation. Proceedings of SIGIR 2000, pp. 184-191, Athens, Greece.

[2] J. W. Murdock, A. Kalyanpur, C. Welty, J. Fan, D. Ferrucci, and D. Gondek, "Type Coercion in Jeopardy!", Full paper under submission to IBM R&D. Extended abstract available.

[3] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. (2010) Building Watson: An Overview of the DeepQA Project. In, *AI Magazine.* Fall, 2010.

[4] James Fan, David Ferrucci, David Gondek, and Aditya Kalyanpur. 2010. Prismatic: Inducing knowledge from a large scale lexicalized relation resource. In Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading, pages 122–127, Los Angeles, California, June. Association for Computational Linguistics.

[5] Jennifer Chu-Carroll, James Fan, Branimir Boguraev, David Carmel, Dafna Sheinwald, and Chris Welty, Finding Needles in the Haystack: Search and Candidate Generation, Full paper under submission to IBM R&D. Extended abstract available.

[6] J. William Murdock, James Fan, Adam Lally, Hideki Shima, and Branimir Boguraev, Textual Evidence Gathering and Analysis, Full paper under submission to IBM R&D. Extended abstract available.

[7] A. Kalyanpur, B. Boguraev, S. Patwardhan, J.W. Murdock, A. Lally, C. Welty, J. Prager, B. Coppola, and A. Fokoue, Structured Data and Inference in DeepQA, Full paper under submission to IBM R&D. Extended abstract available.

[8] C. McCord, J. William Murdock, and Branimir Boguraev "Deep Parsing in Watson" Full paper under submission to IBM R&D. Extended abstract available.

[9] Chang Wang, Aditya Kalyanpur, David Gondek and Branimir K. Boguraev "Relation Extraction and Scoring in DeepQA", Full paper under submission to IBM R&D. Extended abstract available.

[10] Hearst, Marti. (1992) Automatic acquisition of hyponyms from large text corpora. In Proceedings of COLING 1992.

[11] Michele Banko, Michael J Cafarella, Stephen Soderl, Matt Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In In International Joint Conference on Artificial Intelligence, pages 2670–2676.

[12] Thomas Lin, Oren Etzioni, and James Fogarty. 2009. Identifying interesting assertions from the web. In CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management, pages 1787–1790, New York, NY, USA. ACM.

[13] Stephen Soderland, Alan Ritter and Oren Etzioni. 2009. What is this, anyway: Automatic hypernym discovery. In Proceedings of the 2009 AAAI Spring Symposium on Learning by Reading and Learning to Read.

[14] Dekang Lin and Patrick Pantel. 2001. DIRT – discovery of inference rules from text. In In Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 323–328.

[15] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. 2010. Toward an architecture for never-ending language learning. In Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI 2010).

[16] Anselmo Peñas and Eduard Hovy. Semantic Enrichment of Text with Background Knowledge. 1st International Workshop on Formalisms and Methodology for Learning by Reading (FAM-LbR), NAACL 2010

[17] Amsler, Robert. (1980) The structure of the Merriam-Webster Pocket Dictionary. Ph.D. Dissertation, U. Texas at Austin.

[18] Chodorow, Martin, Byrd, Roy and Heidorn, George. (1985) Extracting semantic hierarchies from a large on-line dictionary, In Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics.

[19] Ravichandran, Deepak, and Hovy, Edward. 2002. Learning surface text patterns for a question answering system. In Proceedings of ACL-2002. Philadelphia, PA.

[20] Pantel, Patrick and Pennacchiotti, Marco. (2006) Espresso: leveraging generic patterns for automatically harvesting semantic relations. In Proceedings of COLING-ACL 2006.