Available online at www.sciencedirect.com
ScienceDirect
Journal of Applied Logic 5 (2007) 121–143
JOURNAL OF APPLIED LOGIC
www.elsevier.com/locate/jal

Data-driven type checking in open domain question answering

Stefan Schlobach [a,1], David Ahn [b,2], Maarten de Rijke [b,*,3], Valentin Jijkoun [b,4]

[a] *AI Department, Division of Mathematics and Computer Science, Vrije Universiteit Amsterdam, The Netherlands*
[b] *Informatics Institute, University of Amsterdam, The Netherlands*

Available online 18 January 2006

Abstract

Many open domain question answering systems answer questions by first harvesting a large number of candidate answers, and then picking the most promising one from the list. One criterion for this answer selection is type checking: deciding whether the candidate answer is of the semantic type expected by the question. We define a general strategy for building redundancy-based type checkers, built around the notions of comparison set and scoring method, where the former provide a set of potential answer types and the latter are meant to capture the relation between a candidate answer and an answer type. Our focus is on scoring methods. We discuss nine such methods, provide a detailed experimental comparison and analysis of these methods, and find that the best performing scoring method performs at the same level as knowledge-intensive methods, although our experiments do not reveal a clear-cut answer on the question whether any of the scoring methods we consider should be preferred over the others.

Keywords: Type checking; Question answering; Data-driven methods

1. Introduction

Question answering (QA) is one of several recent attempts to realize information *pinpointing* as a refinement of the traditional document retrieval task. In response to a user's question, a QA system has to return an answer instead of a ranked list of relevant documents from which the user is expected to extract an answer herself. The way in which QA is currently evaluated at the Text REtrieval Conference (TREC, [31]) requires a high degree of precision on the systems' part. Systems have to return *exact answers*: strings of one or more words, usually describing a named entity, that form a complete and non-redundant answer to a given question. This requirement gives QA a strong "high-precision" character. At the same time, however, open domain QA systems have to bridge the potential vocabulary mismatch between a question and its candidate answers. Because of these two aspects, *recall* is a serious challenge for many QA systems.

* Corresponding author.
E-mail address: mdr@science.uva.nl (M. de Rijke).
[1] Partially supported by the Netherlands Organization for Scientific Research (NWO), under project number 220-80-001.
[2] Supported by the Netherlands Organization for Scientific Research (NWO), under project number 612.066.302.
[3] Supported by the Netherlands Organization for Scientific Research (NWO), under project numbers 365-20-005, 612.069.006, 612.000.106, 220-80-001, 612.000.207, 612.066.302, 264-70-050, and 017.001.190.
[4] Supported by the Netherlands Organization for Scientific Research (NWO), under project number 220-80-001.

1570-8683/$ – see front matter
doi:10.1016/j.jal.2005.12.001

To maintain recall at an acceptable level, many QA systems are forced to adopt "non-exact" strategies for many key steps, such as question analysis, retrieval of documents that might contain the answer, and extraction of candidate answers [15,22,24,25]. The underlying assumption is that much of the noise picked up in the early steps can be filtered out in later processing steps. Thus, many QA systems contain a filtering or re-ranking component aimed at promoting correct answers and rejecting or demoting incorrect ones.

In this paper we focus on one particular way of filtering out incorrect answers: *answer type checking*. Here, each question is assigned one or more expected answer types, and candidate answers are discarded if their semantic type is not compatible with the expected answer type(s). Previously, it has been shown that in domains for which rich knowledge sources are available, those sources can be effectively used to perform answer type checking and thus to filter out answers that are wrong because they have an incorrect semantic type [30]—the domain used in that work is the geography domain, where the knowledge sources used include the USGS Geographic Names Information System and the GEOnet Names Server. In other words, in knowledge rich domains, answer type checking has been shown to improve QA performance. In this paper we address the following question: can we generalize answer type checking to domains without rich knowledge sources? More specifically, can we set up a knowledge-poor method for answer type checking whose positive impact on the overall QA performance is comparable to that of knowledge-intensive type checking?

The main contribution of this paper is that we provide positive answers to each of the above research questions. We do so by leveraging the large volume of information available on the web to make decisions about typing relationships between candidate answers and potential answer types. We define a general strategy for building redundancy-based type checkers, built around the notions of *comparison set* and *scoring method*: a comparison set provides a set of types which are related to but sufficiently distinct from an expected answer type for a question to discern correctly typed from incorrectly typed answers; scoring methods are meant to capture the relation between a candidate answer and an answer type. Our focus is on scoring methods; in total, we discuss nine scoring methods, and we find that the best performing scoring method performs at the same level as knowledge-intensive methods, although our experiments do not reveal a clear-cut answer on the question whether any of the scoring methods we consider should be preferred over the others. Different scoring methods result in different behaviors which may be useful in different settings.

The remainder of this paper is organized as follows. We discuss related work in Section 2. Section 3 is devoted to a description of the specific tasks and evaluation measures that we use. We give a high-level overview of our type checking methods in Section 4. Then, in Section 5 we provide a detailed description of the type checkers we have built. Our experimental evaluation, and its outcomes, are described in Section 6. We include an extensive discussion and error analysis in Section 7 before concluding in Section 8.

2. Related work

Many systems participating in the TREC QA track contain an explicit filtering or re-ranking component, and in some cases this involves answer type checking. One of the more successful QA systems, from LCC, has an answer selection process that is very knowledge-intensive [23]. It incorporates first-order theorem proving in attempts to prove candidate answers from text, with feedback loops and sanity-checking, using extensive lexical resources. Closer to the work we report on in this paper is the TREC 2002 system from BBN, which uses a number of constraints to re-rank candidate answers [35]; one of these is checking whether the answer to a location question is of the correct location sub-type. Other systems using knowledge-intensive type checking include those from IBM (which uses the CYC knowledge base [5,26]), the National University of Singapore and the University of Amsterdam (both using external resources such as the Wikipedia online encyclopedia [1,8]), and the University of Edinburgh (which uses a range of symbolic reasoning mechanisms [10]). Some systems take the use of external knowledge sources a step further by relying almost exclusively on such sources for answers and only turning to a text corpus to find justifications for such answers as a final step, if required by a particular QA task [18]. While systems that find their answers externally use many of the same resources as systems that use knowledge-intensive answer type checking, they obviously use them in a different way, not as a filtering mechanism.

Recently, several QA teams have adopted complex architectures involving multiple streams that implement multiple answering strategies [1,5,7,11,16,17]. Here, one can exploit the idea that similar answers coming from different sources are more reliable than those coming from a single source. An answer selection module, therefore, should favor

candidate answers found by multiple streams. In this paper we do not exploit this type of redundancy as a means of filtering or re-ranking; see [1,3,11,16,17] for more work along these lines.

The present paper is intended to specifically evaluate the impact of answer type checking on question answering. The most closely related work in this respect is [30]. In that paper, the utility of knowledge-based type checking using geographical databases for location questions is demonstrated. Building on these findings, Ahn et al. [2] report that extensive filtering results in improvements in accuracy for factoids (going from 42% to 45%), while the accuracy on definition questions drops (from 56% to 48%). As a basis for comparison, we replicate the knowledge-based experiments described in [30] in the present paper; thus, we defer further discussion of them to later sections.

Data-driven ways of combating the problem of noise, in contrast to knowledge-intensive filtering methods, are presented by Magnini et al. [19]. They employ the redundancy of information on the Web to re-rank (rather than filter out) candidate answers found in the collection by using web search engine hit counts for question and answer terms. The idea is to quantitatively estimate the amount of implicit knowledge connecting an answer to a question by measuring the correlation of co-occurrences of the answer and keywords in the question on the web. Schlobach et al. [30] report on disappointing results for re-ranking (instead of filtering) candidate answers using similar co-occurrence measures but between answers and expected answer types rather than answers and questions. We make use of some of the same measures, but we deploy them for filtering by type rather than re-ranking.

3. Task, requirements, and evaluation methods

In this section, we explain the type checking task, lay out the key components that type checking algorithms should provide, and discuss the methodology we use to evaluate type checking methods.

3.1. Type checking candidate answers

Many QA-systems attempt to answer questions against a corpus as follows: for a given question, a number of features, including *expected answer type(s)* (EAT(s)), are extracted. The EATs of the question restrict the admissible answers to specific semantic classes (possibly within a particular domain). For example, within the geographical domain, potential EATs may range over *river*, *country*, *tourist attraction*, etc. Subsequently, documents are retrieved, and from these documents, a list of *candidate answers* is extracted. An *answer selection* process then orders the candidate answers, and the top answer is returned.

If a candidate answer is known *not* to be an instance of any EAT associated with a question, it can immediately be excluded from the answer selection process. When relevant knowledge sources are available, filtering answers by type (or *answer type checking*) is an ideal method to deploy this knowledge in the answer selection process. Briefly, for each candidate answer, one may attempt to extract a *found answer type (FAT)*, i.e., a most specific semantic type of which it is an instance, on the basis of knowledge and data sources. We give a more detailed account of this method in Section 4.2. If the candidate answer's FAT is not compatible with the question's EAT, we reject the candidate answer. We will refer to this approach as *knowledge-intensive type checking* (*KITC*).

Because of the inherent incompleteness of knowledge and data sources available for open domain applications, it may be impossible to determine a FAT for every candidate answer from knowledge resources. Thus, we turn to the web and the sheer volume of information available there as a proxy for engineered knowledge. We take as a starting point the intuition that the typing relation between a candidate answer and a potential answer type may be captured by the co-occurrence of the answer and the type in web documents. Thus, we propose a method in which we assess the likelihood that an answer is of a semantic type by estimating the correlation of the answer and the type in web documents. We will call this the *redundancy-based* approach to type checking (RBTC). Below, we experiment with several correlation measures, and we also look at the web for explicit typing statements (e.g., *VW is a company*) to determine a score for each answer with respect to an answer type. To contain the inherent ambiguity of such a method, we add an ingredient of control by basing our filtering on a comparison of the likelihood of the EAT versus members of a limited set of alternative types, the *comparison set*.

Before discussing the requirements of these methods in more detail, let us briefly mention the limits of our two type checking methods. Obviously, KITC requires data and knowledge sources. For this reason, knowledge-intensive methods are usually restricted to particular domains. In our experiments, this will be the geographical domain. But the

application of redundancy-based methods is also unsuitable for some answer types, such as dates or measurements.[5] In Section 6, we will describe in more detail which answer types we take to be checkable, and which we do not.

3.2. Requirements

The strategies for type checking outlined above require several basic ingredients:

(1) the definition of a set of answer types for consideration, and
(2) a mapping of questions to expected answer types (EATs).

In principal, answer types can be any concept in an ontology of types. As we explain in Section 4.1, we take our (expected) answer types to be WORDNET synsets [21]. Synsets are sets of words with the same intended meaning (in some context). They are hierarchically ordered by the hypernym (more general) and hyponym (more specific) relations. A *sibling* of a type T is a distinct hyponym of a hypernym of T, and the *ancestor (descendant)* relation is the transitive closure of the hypernym relation (the hyponym relation, respectively). We also discuss in Section 4.1 our approach to mapping questions to EATs.

In addition to these basic requirements, knowledge-intensive and redundancy-based approaches each have their own additional distinctive requirements. To develop a knowledge-intensive type checker we have to further introduce:

(3) a method for mapping candidate answers to FATs, and
(4) a notion of compatibility between EATs and FATs (as well as a way of computing compatibility).

In Section 5.1, we discuss several ways of addressing item (3). There, we also formally introduce the notion of compatibility that we employ, which makes use of WORDNET's hierarchical structure.

For redundancy-based type checking, we require two other ingredients, namely:

(5) comparison sets, i.e., sets of types which are related to but sufficiently distinct from an EAT to discern correctly typed from incorrectly typed answers, and
(6) scoring methods to capture the relation between a candidate answer and an answer type.

We address items (5) and (6) in Section 4.3 (in outline) and in Section 5.2 (in detail).

3.3. Evaluation methodologies

We evaluate our answer type checking methods in two ways:

(1) in a *system-dependent* way, by looking at their impact on the overall performance of a particular QA system, and
(2) in a *system-independent* way, by looking specifically at filtering performance on both correct and incorrect answers gathered from multiple systems.

A brief comment on system-independent evaluation: in this type of evaluation, we run our answer type checking methods on so-called *judgment sets* made available by TREC—sets of questions with answers submitted by various TREC participants, together with correctness judgments for the answers. Let us be precise about what we can do with the judgment sets, and how we evaluate success and failure. If we know that an answer is correct, we trivially know that it must be correctly typed. Therefore, none of the correct answers should be rejected, and the quantitative information regarding how many correct answers are rejected is directly relevant. Unfortunately, the situation is not so simple when we consider the incorrect answers in the judgment sets, because there may be incorrect answers that

[5] Candidate answers for date and measurement questions are, in principle, syntactically checkable for type.

Table 1
Evaluation of type checking using judgment sets

	Correct	Incorrect
Rejected	−	?
Accepted	+	?

Table 2
Examples for manually annotated questions

TREC question	Type
1897. What is the name of the airport in Dallas?	`name#n#1 airport#n#1`
1920. When was "Cold Mountain" written?	`date#n#7 writing#n#1`
1958. How fast can a king cobra kill you?	`duration#n#1`
1991. How did Minnesota get its name?	`reason#n#2`
2001. What rock band sang "A Whole Lotta Love"?	`rock_band#n#1`
2049. What president served 2 nonconsecutive terms?	`president#n#3`
2131. Who is the mayor of San Francisco?	`person#n#1`

are correctly typed.[6] Since we are investigating a type checker and not an answer checker, the rejection of a correctly typed incorrect answer should be considered an error.

Table 1 shows what information is directly useful: any correct answer that is rejected is an error on the part of the type checking method. Therefore, a high number of rejected correct answers (i.e., the upper left square of Table 1) indicates poor type checking behavior, while a high number of accepted correct answers (i.e. the lower left square of Table 1), indicates good behavior on the part of a type checker.

Understanding the behavior of a type checking method with respect to incorrect answers is more difficult, as it is possible both that correctly typed incorrect answers are rejected and incorrectly typed incorrect answers are accepted. We have no way of automatically determining the correct types of the incorrect answers, and thus no way to perform proper automatic evaluation of results from the second column. Instead, when we present our experimental results (in Section 6), we give the same acceptance and rejection figures as for correct answers. We also do a proper manual evaluation of a sample of the results for incorrect answers and use the results on the sample to suggest ways of estimating recall and precision of type checking; Section 7 contains a discussion of this manual evaluation.

4. Methods: a high-level overview

Following the requirements set out in Section 3.2, we now provide high-level descriptions of our methods for assigning expected answer types to questions, for knowledge-intensive type checking, and for data-driven checking. Where appropriate, a discussion of specific details of our methods is postponed until Section 5, where we describe in full detail the choices we make in building type checkers.

4.1. Assigning expected answer types

Our question classifier associates each question with one or more EATs. Rather than using an ad hoc answer type hierarchy as many participants in the TREC QA track do [33], we decided to employ WordNet 2.0 noun synsets as the set of possible types. We use supervised machine learning to map questions to EATs. A set of 1371 questions was manually annotated with WordNet noun synsets that best matched the EATs. The annotation guidelines that we used and the annotated set of questions are available at http://ilps.science.uva.nl/Resources/. Table 2 shows examples of manually annotated questions; a word together with a part-of-speech tag and a WordNet sense number (separated by hash marks) uniquely identifies a WordNet synset. Note that in some cases a question type consists of more than one synset (see annotation guidelines for details).

[6] In view of our discussion in Section 2, in which we note that many QA systems perform some sort of type checking, we should expect many incorrect answers to be correctly typed.

The use of WORDNET synsets as question types gives us direct access to the WORDNET hierarchy: e.g., the information that a `president#n#3` is a person (i.e., a hyponym of `person#n#1`) or that `duration#n#1` is a measure.

For classification, each question is automatically tagged for part-of-speech using TreeTagger [32]. The heads of the first and second base noun phrases are then identified (using simple POS-based heuristics), along with other features, such as the first and second words of the question, the main verb, the structure of the first noun phrase (e.g., *X-of-Y*, as in *How many types of human blood...*), presence of several indicative words (*abbreviation*, *capital*, *meaning* etc.). Altogether, 48 features are extracted for every question. Then, we train a memory-based classifier, TiMBL [9], to map feature lists of new questions to EATs, using the manually annotated questions. We also used simple heuristics for word sense disambiguation, e.g., preferring *location* over *organization* over *artifact*. The commonly used "most frequent sense" heuristic yielded similar performance. An informal manual evaluation of the question classifier trained on the set of 1371 annotated questions showed an accuracy of 0.85. We consider this to be a good performance, especially given that the set of all potential question types consists of almost 80,000 noun synsets in WORDNET.

4.2. Knowledge-intensive type checking

While the focus of this paper is on data-driven type checking, for comparison purposes, we replicate experiments involving knowledge-intensive type checking. Briefly, in the knowledge-intensive setting, FATs are determined by looking up, in available knowledge sources, the type information for each candidate answer. As answers may be ambiguous, we often need to associate a number of FATs to each candidate answer. Given the found and expected answer types and a notion of compatibility of two types F and E, the basic algorithm for filtering is as follows:

Extract the expected answer types of each question Q;
for each candidate answer A
  extract the found answer types for A;
  if there is an EAT E of Q and a FAT F of A, such that
    F and E are compatible
    then A is correctly typed;
return the correctly typed candidate answers in the original order;

Fig. 1 shows an ontology[7] with concepts *thing, city*, *state*, *capital* and *river*, where *capital* is more specific than *city*. Furthermore, let the question *What is the biggest city in the world?* have the expected answer type *city*. Assume that *Tokyo* is one of our candidate answers, and that we find in an external data-source that *Tokyo* is of type *capital*. To establish that *Tokyo* is a correctly typed answer we simply have to check that the type *capital* is compatible with, e.g., more specific than type *city*. A different candidate answer *Liffey*, however, which is classified (possibly by a different knowledge source) as a *river*, is rejected as incorrectly typed. In this example, compatibility is calculated based on the hierarchical relations of the simple ontology. In Section 5, we provide an explicit definition of compatibility for our KITC using WORDNET's hypernym and hyponym relations over synsets.

4.3. Redundancy-based type checking

In the knowledge-intensive scenario just described, the type information required for candidate answers is obtained by consulting knowledge sources. Without explicit type information from a knowledge source, we need an alternative way to approximate the determination of FATs for a candidate answer. How can we leverage the large volume of text available on the web to predict (or estimate) typing information for candidate answers?

An important issue emerges in trying to answer this question. It is not obvious how to use the web to determine FATs in a way that is computationally feasible. We cannot simply check a candidate answer against all possible

[7] For simplicity's sake, we introduce small ontologies to explain the basic algorithms. In practice the concept *city* would correspond to the synset `city#n#1`, and the hierarchical relation to a hypernym relation.

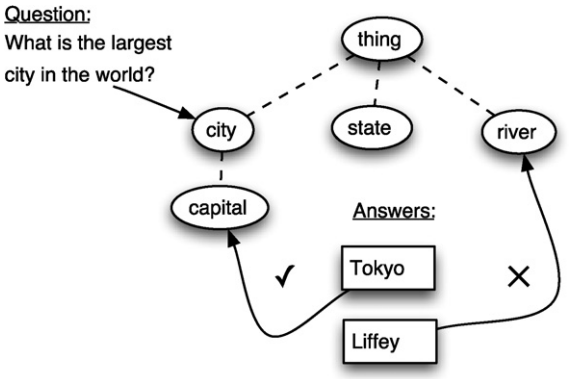

Fig. 1. Knowledge-intensive type checking.

types: since we use WordNet as our source of possible EATs, there would be tens of thousands of possible FATs to check. Even if this were computationally feasible, it would be a bad idea because of potential irrelevant ambiguity of candidate answers [30]. A candidate answer may have incorrectly typed readings that are *a priori* more likely but irrelevant in context. Consider the question *Which motorway links Birmingham and Exeter?* with its correct candidate answer `M5`, which can be linked to many frequent but irrelevant types, such as `telephone_company#n#1` or `hardware#n#3`. In order to ensure practical feasibility and to partially exclude irrelevant types, we only consider for each EAT E a *comparison set* of suitably chosen alternative types. More specifically, given the comparison set $comp(E)$ of an EAT E, and a score $s(A, T)$ indicating the strength of the typing relationship between an answer A and a type T, our algorithm for redundancy-based type checking is as follows:

Extract the expected answer type E of each question Q;
Let $comp(E)$ be the comparison set for E
  for each candidate answer A
    if there is an answer-type T in $comp(E)$ such that
      $s(A, E) \leqslant s(A, T)$
    then A is incorrectly typed
  return the correctly typed candidate answer in the original order;

Fig. 2 provides an illustration of how this generic algorithm might work in a scenario in which the comparison set of an EAT is the set of its WordNet siblings. The question *What company manufactures Sinemet?* is assigned the EAT `company#n#1`. This synset has a number of siblings, which together make up its comparison set; some of them are depicted in the diagram as *medical_institution*, *religion*, and *charity*. Scores are computed for the typing relation between the candidate answer *Parkinson's disease* and each of these potential types—a high score is indicated by a thick arrow; a low score, by a thin arrow. Of course, none of these potential types is an actual type for *Parkinson's disease*, but since the score is based on co-occurrence on the web, the candidate answer has a higher score for *medical_institution* than for *company*, and it is therefore rejected. The other candidate answer, *SmithKlineBeecham*, however, has its maximum score for the EAT *company*, and it is therefore accepted.

The above proposal, then, leaves us with two further notions to define: *comparison set* and *score*. In principle, the choice of comparison sets is arbitrary; what we look for is a number of synsets which are likely to be answer types for the incorrectly typed answers, and unlikely so for the correctly typed answers. As a generic way of creating such comparison sets we suggest using WordNet siblings of the EATs, because we assume that those synsets are sufficiently different from the EAT, while still being semantically close enough to be comparable.

We now turn to the question of how to assign *scores* to typing relations using the web. We experiment with three general approaches to scoring, which we outline below; specific ways of operationalizing these approaches are detailed in Section 5.2.

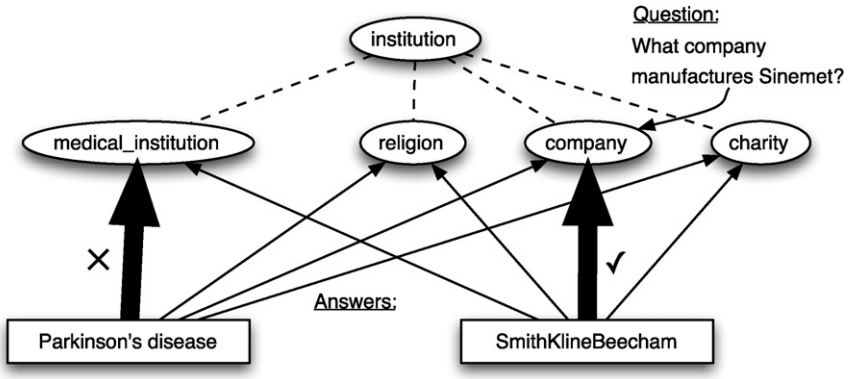

Fig. 2. Redundancy-based type checking.

Typing statements. The assumption here is that the web is large enough that explicit type statements in the form "A is a T" should occur and that the frequency of such type statements should be correlated with their reliability.

Correlation. The assumption here is that a candidate answer should co-occur in the same documents with its actual type more often than with other types. Thus, we use standard co-occurrence measures to estimate the correlation between candidate answers and types. Our underlying assumption is that a web page containing a candidate answer A is a reasonable representation of A and that the occurrence of a potential answer type T on such a web page represents the typing relationship that A is of type T. We can then estimate the likelihood of a candidate answer A having the potential answer type T by collecting co-occurrence statistics for A and T on the web. With these estimates, we can then reject a candidate answer A if it is more likely to be have an alternative answer type T than an EAT E.

Predictive power. The assumption here is that the relationship between an instance and its type is different from the relationship between two terms in the same semantic field. In particular, we expect that the presence of an instance in a document should be highly predictive of the presence of its type, while we have no particular expectation about the predictive power of a type with respect to its instances. The intuition here is that instances, in addition to being referred to by name, are often referred to by descriptions (definite, indefinite, demonstrative) that make use of type terms. For example, we expect that documents that mention the Thames are likely to refer to it not only as *the Thames* but also as *this historic waterway* or *a great big river* or the like (not to mention *the River Thames*). We do not, however, expect that documents that mention a river are particularly likely to mention the Thames. In Section 6, we experiment with several measures in order to quantify this intuition.

5. Building type checkers

To evaluate the general type checking strategies outlined in the previous section, we implemented a number of methods and ran a series of experiments. In this section, we describe the proposed methods in some detail, before discussing the experimental settings in the following section.

5.1. Building a knowledge intensive type checker for geography

To assess the effectiveness of type checking, we implemented type checking for the geography domain and added it to our own QA system, QUARTZ [27]. To implement this method, which we describe in Section 4.2, we need to provide three ingredients, the expected answer types (EATs), a method to extract found answer types (FATs), and a notion of compatibility between types.

We take our answer types from the set of synsets in WORDNET [21]. Synsets are sets of words with the same intended meaning (in some context). They are hierarchically ordered by the hypernym (more general) and hyponym (more specific) relations. A *sibling* of a type T is a hyponym of a hypernym which is different from T. Finally, the *ancestor (descendant)* relation is the transitive closure of the hypernym relation (the hyponym relation, respectively). WORDNET contains many synsets in the geography domain. There are general concepts describing administrative units (e.g., cities or counties) and geological formations (e.g., volcanoes or beaches), but also instances such as particular states or countries. WORDNET provides an easy-to-use ontology of types for answer type checking in the

geography domain. In particular, we benefit in two ways: first, we can make use of the relatively large size of the geography fragment of WordNet, and secondly, we get simple mappings from candidate answers contained in WordNet to usable answer types almost for free.

Our implementation of knowledge-intensive type checking uses WordNet and two publicly available "Name Information Systems" to determine FATs. The Geographic Names Information System (GNIS) contains information "about almost 2 million physical and cultural features throughout the United States and its Territories" [13]. Sorted by state, GNIS contains information about geographic names, including the county, a feature type, geographical location, etc. The GEOnet Names Server (GNS) "is the official repository of foreign place-name decisions ... and contains 3.95 million features with 5.42 million names" [14].

Some candidate answers, such as "Germany" for TREC question *1496, What country is Berlin in?*, occur directly in WordNet. Given our use of WordNet as our ontology of types, we choose all the synsets containing the word "Germany" as FATs. Unfortunately, this case is an exception: many names are not contained in WordNet. The GNS and GNIS databases, however, associate almost every possible geographical name with a geographic feature type. Both GNIS and GNS provide mappings from feature types to natural language descriptions. The FATs determined by the database entries are therefore simply those synsets containing precisely these descriptions. In very few cases, we had to adapt the coding by hand to ensure a mapping to the intended WordNet synsets. A simple example is the GNS type `PPL`, referring to a *populated place*, which we map to the synsets containing the word "city." In the case of complex candidate answers, i.e., answers containing more than one word, the FATs are separately determined for the complex answer string as well as for its constituents using these methods.

We also need a notion of compatibility between answer types. Our definition is based on the WordNet hierarchy: a FAT F is *compatible* with an EAT E, abbreviated by $comp(F, E)$, if F is a hyponym of E, or equal to E.[8]

5.2. Building a redundancy-based type checker

In redundancy-based type checking, we do not calculate FATs but compare the co-occurrence of types and candidate answers on the Web. In order to do this, we need a scoring mechanism to link answers to answer types. Furthermore, to contain the possible semantic ambiguity of the candidate answers, we need to restrict the comparison of scores to a suitable comparison set.

5.2.1. The comparison set

At first glance, it is not obvious how to use a scoring mechanism, which simply provides a numerical measurement to link candidate answers and answer types, for filtering. The fact that a candidate answer and an expected answer type (EAT) are given a particular score tells us little by itself. One possibility, investigated in [30], is to re-rank candidate answers according to their score for EATs. In this paper, we pursue an alternative line: instead of comparing the score for a given candidate answer and the EAT with the scores of other candidate answers and the EAT, we compare, for each candidate answer, its score for the EAT with its score for other answer types. In Section 4.3, we gave a schematic description of this redundancy-based type checking method, in which a candidate answer is accepted if the score of the answer and the EAT is higher than the score of the answer with any of the answer types in a *comparison set*. How to construct this comparison set in a generic way is now the obvious question. A comparison set for an EAT should ideally contain, for each incorrectly typed answer A, at least one type which is more likely to be the answer type of A than the expected answer type, but, for correctly typed answers, no such type.

The general idea can best be explained with a simple example. Consider the following question: *What dessert is made with poached peach halves, vanilla ice cream, and raspberry sauce?* with EAT `dessert#n#1`, and candidate answers *salmon with tangy mustard*, *alcoholic beverage* and *Peach Melba*. Ideally, our type checker will accept the last answer, and reject the first two. But how is it supposed to do so? For our implementation of redundancy-based filtering, we benefit from the fact that our answer types are WordNet synsets—we can use (a particular subset of) the siblings of an EAT type as its comparison set. For the above example, the siblings of `dessert#n#1` are `appetizer#n#1`,

[8] The notion of compatibility can be more complex depending on the ontology, the available reasoning mechanisms, and the available data sources. If we consider more complex answer types, such as conjunctive ones, we may have to relax the definition. For example, consider a complex EAT *river & German & muddy*. In this case, an answer with a FAT *German & river* could be considered compatible to the EAT even though the FAT is not more specific than the EAT.

Table 3
Answer types and their respective comparison sets

Answer type	Comparison set
college#n#1	sector administration corps school university colony leadership opposition jury
company#n#1	charity religion academy medical_institution
continent#n#1	∅
country#n#1	reservation municipality
location#n#1	thing object substance thing enclosure sky
person#n#1	animal plant microorganism anaerobe hybrid parasite host individual mutant stander agent operator danger
president#n#3	sovereign
river#n#1	branch brook headstream

entree#n#1, pudding#n#1. Intuitively, the first answer *Salmon with tangy mustard* is more likely to be an appetizer than a dessert, while the second, *Alcoholic beverage*, is more likely to be related to an entree. Only the third answer is more likely to be a dessert than any of the other types. Our assumption is that the WORDNET siblings of an EAT are conceptually similar to it, while being sufficiently distinct to discern incorrectly typed answers.

To make this method generally workable in practice, we need to adapt it slightly. First, the number of siblings in WORDNET might be huge; for example, a large number of known countries are listed as siblings of answer type country#n#3. For this reason, we exclude named entities and leaves from the comparison sets and further exclude some (hand-picked) stop-types, such as power#n#1, self#n#1, and future#n#1. To give an intuition of what comparison sets actually look like, Table 3 provides a list of comparison types for eight answer types which we will study in more detail in our manual evaluation in Section 7. For better readability, we only give one informal label for each synset in the comparison sets.

Now that we have established what our comparison sets are, we define a number of scoring methods for the typing relation between answer types and candidate answers.

5.2.2. Scoring typing relations

As we discussed in Section 4.3, we experiment with three different kinds of scores for the typing relation between a candidate answer and a potential type. We now describe in more detail the actual measures we use in our experiments.

All of our scores are based on hit counts of terms on the Web. More precisely, if t is an arbitrary natural language term, $hit(t)$ denotes the number of web pages containing t, as estimated by Google. We use the $+$ operator to denote co-occurrence of terms, i.e., $hit(t_1 + t_2)$ denotes the number of web pages containing both t_1 and t_2. We will combine strings into terms using quotes, and use the "*" as a placeholder for a single word. Finally, let N be the total number of web pages indexed by Google.

5.2.2.1. Explicit type statements We compute two different scores using hit counts of explicit type statements on the web.

Strict type statement occurrence (*STO*). This measure simply looks on the web for occurrences of strict type statement of the form "A is a(n) T", where A is a candidate answer and T is a type term. Thus,

$$STO(T, A) = \frac{hc(\text{"A is a(n) T"})}{N}$$

is the estimated probability that a web document contains either the sentence "A is a T" (when T begins with a consonant) or "A is an T" (when T begins with a vowel).

Lenient type statement occurrence (*LTO*). This measure is just like STO, except that the type statements are liberalized to allow for: (1) present or past tense statements and (2) one or two arbitrary words before the type term. Thus,

$$LTO(T, A) = \frac{hc(\text{"A (is | was) a(n) * T"}) + hc(\text{"A (is | was) a(n) * * T"})}{N}$$

is the estimated probability that a web document contains a sentence matching one of the 6 patterns "A is a * T", "A is an * T" "A was a * T", "A was an * T" "A is a * * T", "A is an * * T" and "A was a * * T", "A was an * * T" (again, the "a" patterns are for type terms beginning with consonants, and the "an" patterns are for type terms beginning with vowels).

5.2.2.2. Phrase co-occurrence We compute two scores that measure the degree of correlation between two co-occurring words or phrases. Both measures are also used by [19] in their work on validating answers using redundancy of information on the web.

Pointwise mutual information (*PMI*). Pointwise mutual information [6] is a measure of the reduction of uncertainty that one term yields for the other.

$$PMI(T, A) = \log \frac{P(T, A)}{P(T) \times P(A)}.$$

We use hit counts to generate maximum likelihood estimates for the probabilities:

$$PMI(T, A) = \log \frac{hc(T + A) \times N}{hc(T) \times hc(A)}.$$

In general, since we are only comparing these scores with respect to their relative rank and not their magnitudes and since we further only compare different choices of T for any given choice of A, we can simplify this to:

$$PMI\_score(T, A) = \frac{hc(T + A)}{hc(T)}.$$

Log-likelihood ratio (*LLR*). As [19] point out, there has been some discussion in the literature regarding the inadequacy of PMI as a correlation measure for word co-occurrence in the face of sparse data (see [20, Chapter 5] for details). Since we, like them, use web hit counts not only of individual words but also of entire phrases, sparse data is indeed a potential problem, and we follow them in using Dunning's log-likelihood ratio as an alternative correlation measure [12]. In general, the log-likelihood ratio can be used to determine the degree of dependence of two random variables; Dunning applies it to the problem of finding collocations in a corpus (in preference to the t-test or the χ^2-measure). Consider

$$\lambda = \frac{L(H_1)}{L(H_2)},$$

where H_1 is the hypothesis of independence, H_2 is the hypothesis of dependence, and L is the appropriate likelihood function. Since we can think of the occurrence of a term in a document as a Bernoulli random variable, we can assume a binomial distribution for the occurrence of a term across the documents sampled. Thus, the likelihood function is:

$$L = \binom{n_1}{k_1} p_1^{k_1} (1 - p_1)^{n_1 - k_1} \times \binom{n_2}{k_2} p_2^{k_2} (1 - p_2)^{n_2 - k_2},$$

where p_1 is the probability of the second term occurring in the presence of the first ($P(T2|T1)$) and p_2 is the probability of the second term occurring in the absence of the first ($P(T2|\neg T1)$). The first multiplicand, then, is the probability of seeing k_1 documents containing both terms out of n_1 documents containing the first term, while the second multiplicand is the probability of seeing k_2 documents containing the second term out of n_2 documents not containing the first term.

Turning back to the likelihood ratio, H_1 is the hypothesis of independence, thus, that $p_1 = p_2 = P(T2) = \frac{k_1 + k_2}{n_1 + n_2}$, which is just the maximum likelihood estimate of the probability of $T2$. The hypothesis of dependence, H_2, is that p_1 is the maximum likelihood estimate of the probability $P(T2|T1) = \frac{k_1}{n_1}$ and that p_2 is the maximum likelihood estimate of the probability $P(T2|\neg T1) = \frac{k_2}{n_2}$.

In order to scale this ratio to make comparison possible, we use the log-likelihood form $-2 \log \lambda$. Thus, for our particular situation, we compute

$$LLR(A, T) = 2\big[\log L(p_1, k_1, n_1) + \log L(p_2, k_2, n_2) - \log L(p, k_1, n_1) - \log L(p0, k_2, n_2)\big],$$

where

$$\log L(p, n, k) = k \log p + (n - k) \log(1 - p)$$

and

$$k_1 = hc(T + A), \qquad k_2 = hc(T) - hc(T + A), \qquad n_1 = hc(A), \qquad n_2 = N - hc(A)$$

and

$$p_1 = \frac{k1}{n1}, \qquad p_2 = \frac{k2}{n2}, \qquad p_0 = \frac{hc(T)}{N}.$$

5.2.2.3. Predicting types from answers We compute three scores that attempt to estimate the dependence of occurrences of types on the occurrence of candidate answers.

Conditional type probability (*CTP*). This is the most basic measure of how the occurrence of an answer affects the occurrence of a type.

$$CTP(T, A) = P(T|A) = \frac{P(T, A)}{P(A)}.$$

As usual, we estimate the probabilities with maximum likelihood estimates based on hit counts:

$$CTP(T, A) = \frac{hc(T + A)}{hc(A)}.$$

And, again, since we only use this score to compare different values of T for any given value of A, we need only look at:

$$CTP\_score(T, A) = hc(T + A).$$

Corrected conditional probability (*CCP*). CTP is biased toward frequently occurring types; [19] introduce CCP as an ad hoc measure in an attempt to correct this bias:

$$CCP(T, A) = \frac{P(T|A)}{P(T)^{2/3}}.$$

As usual, we estimate the probabilities with maximum likelihood estimates and ignore factors that are constant across our comparison sets:

$$CCP\_score(T, A) = \frac{hc(T + A)}{hc(T)^{2/3}}.$$

Information gain (*IG*). Information gain is the amount of information about a hypothesis that we gain by making an observation. Alternatively, we can think of information gain as the reduction in the amount of information necessary to confirm hypothesis yielded by an observation. One common application of IG is in feature selection for decision trees [28]. We use an odds-ratio-based formulation of information gain [4]. Consider

$$IG(T, A) = -\log \frac{P(T)}{P(\neg T)} - -\log \frac{P(T|A)}{P(\neg T|A)}.$$

We can interpret $-\log \frac{P(T)}{P(\neg T)}$ as quantifying the prior information needed to confirm T, while $-\log \frac{P(T|A)}{P(\neg T|A)}$ quantifies the information needed after observing A. $IG(T, A)$, then, specifies how much information is gained about T as a result of seeing A.

We again generate maximum likelihood estimates for the probabilities using hit counts, so that we actually compute

$$IG\_score(T, A) = \log hc(T + A) + \log(N - hc(T)) - \log hc(T) - \log\big(hc(A) - hc(T + A)\big).$$

6. Experiments and results

In the previous section, we explained the choices we made in building two different kinds of type checkers. Knowledge-intensive type checking (KITC) was initially introduced for the geography domain, whereas redundancy-based type checking (RBTC) is, in principle, applicable to open domain QA. To evaluate type checking in general, and these two frameworks in particular, we performed a number of experiments. For the sake of completeness, we begin in Section 6.1 by restating previous results on adding KITC to a particular QA system to type check answers in the geography domain [30]. Since the focus of this paper is on redundancy-based methods, we ran some of the same experiments using RBTC. The results of these experiments are also described in Section 6.1.

In order to evaluate the specific contribution of type checking to QA, we also performed several sets of system-independent experiments. The first set of these experiments, described in Section 6.2, are still restricted to the geography domain. Since the goal of introducing RBTC is to extend type checking to open domain QA, however, our other set of system-independent experiments, described in Section 6.3, applies type checking to open domain questions.

6.1. Type checking for question answering in a closed domain: geography

One of the goals of our experiments with type checking is a proof of concept, i.e., evaluating whether type checking can actually be successfully integrated into a QA system. In this section, we discuss our experience with domain-specific type checking in a system-dependent setting. Particular focus is placed on two issues. First, we want to find out whether type checking improves our question answering performance. Second, what are the advantages of each of the two strategies—knowledge-intensive and redundancy-based? To address the first issue, we applied KITC in the geography domain [30].

Experiments with KITC. We evaluate the output of our QA system on 261 location questions from previous TREC QA topics and on 578 location questions from an on-line trivia collection [29], both without and with type checking. These questions are fed to our own QUARTZ system [27], and the list of candidate answers returned by QUARTZ is subjected to answer type checking. We use two evaluation measures: the mean reciprocal rank (MRR: the mean over all questions of the reciprocal of the rank of the highest-ranked correct answer, if any) [34], and the overall accuracy (percentage of questions for which the highest-ranked answer returned is correct).

For 594 of the 839 questions, we determined over 40 different expected answer types. The remaining questions either did not have a clear answer type (questions such as *Where is Boston?*), or were not real geography questions (such as *What is the color of the German flag?*). The types *country*, *city*, *capital*, *state*, and *river* were the most common and accounted for over 60% of the questions. In these experiments, we did not use the generic question type extraction method described in Section 4.1, as this was not available at the time. Instead, we used a simple pattern-matching approach, which we fine-tuned to the geographical domain for high coverage of typical EATs. We evaluated the EAT extraction process by hand and found an accuracy of over 90%.

To establish a realistic upper bound on the performance of answer type checking, we manually filtered incorrectly typed candidate answers for all the questions in order to see how much *human answer type checking* would improve the results. Then, we turned to our automatic, knowledge-intensive methods. To analyze the influence of the use of databases on type checking, we ran both the algorithm described in the Section 4.2 and a version using only WORDNET to find the FATs. This latter method is denoted by `KITC_WN` below, while the full version is denoted as `KITC_WN&DB`.

Results. Table 4 summarizes the main results of our experiments. For each of the methods, we give the total number of correct answers (with percent-improvement over no type checking in parentheses) and the corresponding accuracy, as well as the MRR. Note that in these experiments we consider inexact answers to be correct.

These quantitative results show that type checking is useful, and that it can successfully be applied to question answering. Knowledge-intensive type checking can substantially improve the overall performance of a QA system for geography questions, although the best available strategy performs substantially worse than a human. What is surprising is that using the GNIS and GNS database for type checking actually leads to a substantial drop in performance compared to type checking with WORDNET alone. We do not repeat the qualitative assessment of these experiments here, but refer to [30] for a detailed analysis.

Table 4
Results for system-dependent, closed-domain type checking (Section 6.1)

Strategy	Correct answers	Accuracy	MRR
No type checking	244	29.0%	0.33
Human type checking	331 (+36%)	36.4%	n/a
`KITC_WN&DB`	271 (+11%)	32.3%	0.37
`KITC_WN`	292 (+20%)	34.8%	0.38

Table 5
Results for system-dependent, closed-domain type checking, exact evaluation (Section 6.1)

Strategy	Correct answers (percent-improvement)
No type-checking	69
KITC_WN	87 (+26.0%)
RBTC_IG	74 (+7.3%)
RBTC_STO	83 (+20.3%)

Comparing RBTC and KITC on QUARTZ. So far so good: the experiments show that type checking works. We would like to see, then, whether RBTC can yield similar improvements in QA performance. To this end, we applied RBTC in the same way as we did KITC in the experiments described above. These experiments with QUARTZ were performed on a 2004 version of our QUARTZ system that takes advantage of inexact evaluation to produce relatively long candidate answers that contain, but are not limited to, actual answers.[9] Redundancy-based methods, however, can not easily be used to type check such long candidate answers, as these methods usually take the entire candidate answer string into account and cannot simply ignore the imprecise part of the answer. This fact is one of the reasons for the failure of the redundancy-based type checking methods described in [30]. Note that this difference is also due to the set-up of RBTC and KITC, as we check types of substrings in the latter, but, for efficiency reasons, not in the former.

In order to facilitate at least a limited system-dependent comparison between the two type checking frameworks described in this paper, we ran type checking experiments using two redundancy-based methods (RBTC_IG and RBTC_STO, i.e., redundancy-based type checking with the information gain and strict type statement occurrence measures) and knowledge-intensive type checking with WORDNET only, but we used *exact* evaluation. The experiments were performed on the same data set of 839 questions described above; see Table 5. Without type checking, only 69 of the 839 questions have an *exact* correct answer as the first returned answer, which is less than 10%. All three type checking methods increase the number of correct answers substantially, with knowledge intensive type checking with WORDNET helping the most, increasing the number of correct answers by 26%.

Even though KITC outperforms the two redundancy-based methods, the results for RBTC are also promising. There is a clear improvement of the overall performance of the system, particularly when applying RBTC with the STO measure. However, as we only compare exact answers within a particular question answering system, it is difficult to know whether these results generalize, which is why we turn to a different type of evaluation.

6.2. Comparing KITC to RBTC in the geography domain

System-independent evaluation. The purpose of the set of experiments described in this subsection is to investigate on a broader scale how well a redundancy-based type checking method compares to a knowledge-intensive one on a closed domain. In the previous section, we note that limitations of our QA system make it difficult to do a large-scale comparison of knowledge-intensive and redundancy-based type checking methods. Thus, in this set of experiments, we use a system-independent methodology, as discussed in Section 3.3.

The data. For a generic comparison of the two type checking paradigms, we need candidate answers from a variety of QA systems. Fortunately, TREC provides the judgment sets from previous evaluation exercises: the judgment sets for TREC 11 and TREC 12 comprise 1000 questions with 25,788 candidate answers, which are judged to be correct[10] or incorrect. We use these judgment sets as the basis of our system-independent evaluation.

Types that can be checked by both KITC and RBTC. In order to compare knowledge-intensive and redundancy-based methods, we have to have a level playing field, i.e., we can only compare the two frameworks on questions that can actually be checked using both methods. In our case, there are 133 questions (with 34 different answer-types) that can be checked both using KITC and RBTC. In particular, we restrict our experiments to questions that our knowledge-intensive type checker can handle; these are the hyponyms of the synsets:

[9] In inexact evaluation, an answer returned by a system is judged as correct just as long as it contains the correct answer as a substring.

[10] For the purposes of our evaluation, it does not matter whether an answer is justified or not. Therefore, we count unjustified answers as correct answers.

Table 6
Redundancy-based versus knowledge-intensive filtering in a closed domain (Section 6.2)

	No TC	PMI	LLR	CTP	CCP	IG	STO	LTO	KITC_ WN	KITC_ WN&DB
CA acc.	1082	501	876	890	902	852	951	983	872	943
(% acc.)		46%	80%	82%	83%	78%	87%	90%	80%	87%
IA acc.	2906	1115	1974	1983	1864	1778	1908	1971	1487	1920
(% acc.)		38%	67%	68%	64%	61%	65%	67%	51%	66%

- location#n#1,
- body_of_water#n#1,
- color#n#1,
- institution#n#1,
- land#n#3.

These answer-types were hand-picked for several reasons: not only do we expect them to lend themselves to knowledge-intensive methods because of the type of information modeled in the particular fragment of WordNet, but we also take into account what sort of data is available in WordNet. In this case, we know that WordNet has exhaustive data on geographic information and colors, but little on persons. Nevertheless, the above restrictions are relatively arbitrary.

In comparing the two frameworks, we use several variants of each. For KITC, we vary the knowledge sources: one run uses WordNet as the only data source, while the second also uses the two geographical databases GNIS and GNS. We denote the first method as KITC_WN and the second one as KITC_WN&DB. For RBTC, we vary the scoring method, using each of the scores described in Section 5.2: PMI, LLR, CTP, CCP, IG, STO, and LTO.

Results. On the 133 questions that are checkable by both RBTC and KITC methods we applied a total of nine different algorithms. Table 6 shows the outcomes, with the first and second rows indicating the number and percentage of correct answers accepted and the third and fourth rows indicating the number and percent of incorrect answers accepted. In total there were 3988 answers to the questions, of which 2906 were incorrect and 1082, correct. Each algorithm has its own column, with the first column indicating the result of doing no type checking at all.

As we discuss in Section 3.3, making sense of these results is not entirely straightforward. Looking first at the first two rows, where we can be relatively confident of an interpretation, we see that the methods that make use of explicit type statements from the web (STO and LTO) do as well as or better than the knowledge-intensive methods. Furthermore, we see that while PMI performs quite poorly (supporting the hypothesis that it is an inappropriate measure for phrase correlation on the web), the other symmetric correlation measure, LLR, performs about as well as the measures that try to quantify the intuition that the occurrence of type terms is more dependent on the occurrence of instance terms than vice versa.

Turning to the second two rows, we must reiterate that we cannot be certain that a high rejection rate (i.e., low acceptance rate) for incorrect answers is an entirely good thing. Nevertheless under the assumptions that at least a reasonable proportion of incorrect answers are incorrectly typed and that acceptance rates of correctly typed incorrect answers are similar to those of correct answers, we can take lower acceptance rates for incorrect answers to be a more or less positive sign. As we will see in Section 7, these assumptions seem to be supported by a restricted manual evaluation.

That said, we see that the explicit type statement measures, while having the highest acceptance rates for correct answers (a good thing), also have acceptance rates for incorrect answers comparable to, i.e., as low as, most of the other methods. In particular, they compare quite favorably to the knowledge-intensive method using geographical databases (KITC_WN&DB). Of the co-occurrence-based measures, only IG has a substantially lower acceptance rate for incorrect answers (coupled with a reasonable acceptance rate for correct answers, unlike PMI), although CCP also has a reasonable balance of a relatively high acceptance rate for correct answers and a relatively low acceptance rate for incorrect answers. Also, note that of the methods with reasonable acceptance rates for correct answers, the knowledge-intensive method using only WordNet has by far the lowest acceptance rate for incorrect answers.

Table 7
Open domain type checking (Section 6.3)

	No TC	PMI	LLR	CTP	CCP	IG	STO	LTO	KITC_WN
CA acc.	2342	955	1736	1767	1732	1517	1625	1824	2004
(% acc.)		40%	74%	75%	73%	64%	69%	77%	85%
IA acc.	7954	2581	4885	4872	4543	3864	3966	4268	5992
(% acc.)		32%	61%	61%	57%	48%	49%	53%	75%

Finally, a word of caution is in order. We will see in the following section that the results for RBTC significantly worsen once we consider questions from other domains. This indicates that the setup of our experiments for the comparison of knowledge-based versus redundancy-based methods favors simple questions (and answers). At this moment we do not know whether these more simple questions lend themselves more easily to knowledge intensive rather than redundancy based type checking, which could explain the above results.

6.3. Type checking in open domains

In the previous section, we demonstrate that redundancy-based type checking methods can achieve performance similar to that of knowledge-intensive methods in restricted domains. In this section, we would like to see whether this level of performance can be maintained when we move to an open-domain setting. First, though, we must discuss what we mean by an open-domain setting.

Checkable questions. As mentioned earlier, not all questions are amenable to the type checking methods we present in this paper; in particular, date and measurement questions are far better suited to syntactic type checking methods. Here, we consider answer-types that are hyponyms of the following synsets:

- body_of_water#n#1
- craft#n#1
- activity#n#1
- act#n#2
- perception#n#3
- land#n#3
- color#n#1
- social_group#n#1
- artifact#n#1
- event#n#1
- person#n#1
- creation#n#2
- substance#n#1
- spiritual_being#n#1
- unit#n#1
- location#n#1

Of the 1000 questions in the TREC judgment sets we consider, 350 are checkable with 159 different types.

Results. For this set of experiments, we performed type checking on the 350 checkable questions (out of the 1000 questions in the judgment sets of TREC 11 and 12) using all seven redundancy-based measures. For comparison purposes, we also performed type checking using the knowledge-intensive method with WordNet only. Note, of course, that this method can only be applied to a subset of these 350 questions; for the other questions, all answers are accepted. In total, there were 25,788 candidate answers, of which 10,292 belonged to checkable questions. The results of these experiments are given in Table 7.

Here, we see again that PMI performs abysmally, while LTO performs quite well, with the highest acceptance rate for correct answers (apart from the knowledge-intensive method) and among the lowest acceptance rates for incorrect answers. We see further that IG again has the lowest acceptance rate for incorrect answers (among those methods with reasonable acceptance rates for correct answers, i.e., excluding PMI), and that LLR, CTP, and CCP perform similarly. In a departure from the closed-domain experiments, the other explicit type statement method, STO, has lower acceptance rates than most of the methods. This difference between the two explicit type statement methods may indicate that relatively well-defined domains such as geography simply lend themselves better to simple type statements than other domains. A simple explanation could be that our STO are in present tense, whereas the LTO patterns also match for past references. Given that statements about geography are mostly in present tense, whereas typing statements in other domains are more likely to refer to the past, this could explain the differences. Also, as we point out above, the knowledge-intensive method, KITC_WN has high acceptance rates for both correct and incorrect answers because it cannot type check answers for many of the questions under evaluation.

In order to get a better grip on what these numbers are telling us, we turn, in Section 7, to a manual evaluation of a limited sample of the data, from which we can try to extrapolate more useful recall and precision measures for our type checking methods.

7. Discussion and error analysis

The shortcomings and problems of knowledge-intensive type checking are exhaustively discussed in [30]. What remains is to do an analysis, quantitative as well as qualitative, of the outcome of the experiments with our new redundancy-based type checking methods and to discuss the lessons that we have learned from it.

7.1. What do our numbers really mean?

As discussed in Section 3.3, it is not immediately clear how to interpret our quantitative results. Remember that we have two types of candidate answers for each question: correct and incorrect. It is clear that rejecting any correct answer is an error; in other words, all correct answers should be accepted. This means that the acceptance rate given in second row in Tables 6 and 7, which indicates the percentage of correct answers that are actually accepted by each method, can be interpreted as a recall measure, at least for correct answers.

But what can be said about the incorrect answers that are rejected or accepted by type checking? To what extent can we expect that the rejected answers are indeed incorrectly typed, or that the accepted answers are correctly typed? Or, in the terminology of recall and precision, what can we say about the recall for incorrect answers (i.e., the proportion of rejected incorrect answers that are, in fact, incorrectly typed, or, equivalently, the proportion of accepted incorrect answers that are correctly typed)? What can we say about precision at all (i.e., the proportion of accepted answers, correct or incorrect, that are actually correctly typed)?

We have argued that the acceptance rate for correct answers is essentially a recall measure for correct answers. If we assume that the behavior of our type checking methods is independent of the correctness of the answers, we would expect the acceptance rate of correctly typed incorrect answers to be similar to that of correct answers. Based on this assumption, then, the acceptance rate of correct answers can be viewed as a plausible estimate for overall recall. Furthermore, if we could estimate the proportion of incorrect answers that are, in fact, correctly typed, we could combine that estimate with this assumption to estimate precision, as well.

In order to test this assumption that acceptance rates for correct answers and for correctly typed incorrect answers are similar, and also to get a sense of the proportion of incorrect answers that are correctly typed, we performed a manual evaluation on the most frequent answer-types (those with more than 150 candidate answers each). We chose these types because they are hopefully less dependent on particular questions and answers. We look in particular at the results of the information gain measure `IG` on these eight types. Information gain yields the highest rejection rates for incorrect answers and should thus offer the greatest possibility for false positives and negatives among the incorrect answers. The breakdown of acceptance rates for correct and incorrect answers is given in Table 8. Note that the overall acceptance rate for correct answers to questions of one of these eight types is 73%.

For the 3188 incorrect candidate answers, we verified by hand the correctness of the type checking algorithm. The results of this manual evaluation are given in Table 9. Note that the total number of incorrect answers does not add up

Table 8
Results of `IG` on the 8 answer-types with the most candidate answers

EAT	Accepted incorrect answers	Accepted correct answers
`college#n#1`	70 of 134 (52%)	1 of 21 (4%)
`company#n#1`	86 of 116 (74%)	35 of 36 (97%)
`continent#n#1`	104 of 108 (96%)	86 of 89 (96%)
`country#n#1`	586 of 643 (91%)	342 of 356 (96%)
`location#n#1`	622 of 1005 (61%)	302 of 384 (78%)
`person#n#1`	217 of 820 (26%)	68 of 247 (27%)
`president#n#3`	207 of 251 (82%)	70 of 74 (94%)
`river#n#1`	59 of 111 (53%)	18 of 48 (37%)
Σ	1951 of 3188 (61%)	922 of 1255 (73%)

Table 9
Evaluating assessment measures

Correct	Correctly typed	Total	Accepted	Rejected
		1255	922 (73%)	333 (27%)
Incorrect	Correctly typed	1741	1242 (71%)	499 (29%)
	Incorrectly typed	1396	694 (50%)	702 (50%)

to the 3188 figure given in Table 8. This is because the human evaluators lacked enough domain knowledge to assign a type to some of the candidate answers.

From these numbers, we can compute recall and precision for our sample. Recall is the ratio of accepted correctly typed answers to all correctly typed answers: 72%. Precision is the ratio of accepted correctly typed answers to all accepted answers: 76%. More interestingly, we can also use these results to suggest extrapolated estimates of recall and precision for the entire set of answers.

First of all, the results confirm our hypothesis that acceptance rates for correctly typed incorrect answers should be similar to acceptance rates for correct answers. Thus, we can tentatively suggest that acceptance rates for correct answers might, in general, be reasonable estimates for overall recall of our type checking methods. We further see from these results that more than 55% of the incorrect answers are correctly typed (confirming our earlier suggestion that, given that these answers are generated by systems, many of which perform their own answer type checking, we should see a bias, even among incorrect answers, toward being correctly typed). If we assume that this ratio of correctly to incorrectly typed incorrect answers holds for the judgment sets, in general, we can use our estimated recall (in this case, for the `IG` method, 73%) and the known acceptance rate for incorrect answers (in this case, 61%) to compute an estimated precision score of 75%. Note that if we estimate precision in this way, precision does, in fact, increase as the acceptance rate of incorrect answers decreases, just as we suggested earlier.

Of course, there are several caveats we should note regarding our sample. First of all, the acceptance rate (for the `IG` method) for correct answers in this sample (73%) is 9 percentage points higher than the acceptance rate for the `IG` method across all correct answers (64%). In other words, our sample is biased towards types that are, in a sense, easy to verify, which may have an impact on the independence of type checkability from correctness. Secondly, the sample contains a higher proportion of correct answers (28%) than the judgment sets at large (22%), which may be related to the high proportion of correctly typed incorrect answers. Nevertheless, this small manual evaluation does suggest possible interpretations of the acceptance rates for correct and incorrect answers that we give in Tables 6 and 7 in the previous section.

7.2. What goes wrong, and why?

To determine what goes wrong with type checking (and ultimately, why), we have to go beyond the quantitative analysis of the results presented in the previous section in Table 7. In addition to those quantitative measures, we have to look at specific examples to see why correctly typed answers are rejected, and why we accept incorrectly typed answers. For this analysis, we looked at the results of the two methods `IG` and `STO`, as they involve two very different scoring mechanisms, and as they had the widest range between accepted incorrect answers and accepted correct answers.

7.2.1. False negatives: rejecting a correctly typed answer

For all of the meaningful redundancy-based type checking measures, Table 7 shows a relatively high number (between 15% and 36%) of rejected correct (and therefore correctly typed) answers. This is cause for concern, and we looked at a number of cases individually. There are different types of errors, often related to the *semantic ambiguity of the data*, i.e., the answer or the answer types, shortcomings of the *scoring mechanisms* and the *counting method* itself.

Issues related to the counting method. Redundancy-based type checking as described in this paper is based on the assumption that there is likely to be a correlation between the occurrence of an answer and its semantic type in documents on the Web. This, although quite often the case, is not universally true. There are cases where the direct link between the semantic type and the descriptive term is rather weak. This is usually the case for generic answer types

Table 10
VW: company or religion?

hc("VW" + "company") = 250,000	hc("VW") = 2,490,000
hc("VW" + "religion") = 98,800	hc("company") = 9,910,000
hc("religion") = 3,580,000	
IG("religion",VW) = 1.96	IG("company",VW) = 1.95

such as building, location or person.[11] Quantitatively, this can be seen in the relatively high number of rejected correct answers for questions of these types (for example, 43% instead of 36% for these three types using the information gain measure). A good example is the answer *northern Greece* to a question with answer type *location*. Here, the answer was erroneously rejected because it was judged more likely to be of type *sky*. It is not surprising that the semantic link between the answer and this particular element of the comparison set is stronger than with the EAT `location#n#1`. Even for more specific types, there are examples where the syntactic link between the answer and its type is too weak. *Strawberries*, for example, are linked more strongly to the alternative type `can#n#1` than to the EAT `dish#n#1`.

On a more technical note, there were also direct problems with the hit counts using Google. Because the hit counts for the term "President Clinton" is erroneously higher than for the term "President," a useful comparison based on these numbers is impossible.[12]

Issues related to the scoring mechanisms. Even if both the hit counts and the link between syntax and semantics line up in our favor, the scoring mechanisms are not always adequate. The most simple example is when type checking is done using scores based on strict phrase occurrence. Remember that this method only counts hit counts for phrases "A is a(n) T" for an answer A and a type T. Unfortunately, these explicit typing statements are often missing, even if one would expect otherwise. For example, the phrase "Nelson Mandela is a president" was not detected on the Web, and consequently Nelson Mandela was rejected as a president by `STO`. A simple solution for this is to extend the set of patterns, and with this the likelihood of finding perfect matches. Although this is a promising step further it has the disadvantage of increasing the number of necessary search engine queries, and therefore significantly slows down the type checking process.

Another issue is the treatment of rare versus frequent answer types. Here, we can see a real difference between the measures introduced in Section 5.2. For example, the IG measure yields a higher score for *Nile* with *head-stream* than with *river*, while the LLR measure does not. On a positive note, it is nice to see that using the CCP measure, it is even possible to distinguish head-streams and rivers, as the White Nile is correctly recognized as a head-stream, and the Nile as a river.

Data related issues. The advantage of the previous example is that both the types and the candidate answers are unambiguous, which is usually not the case. Based on the IG measure, *VW* is more likely to be a religion than a company. Studying the raw numbers, it is easily seen that this really makes sense; see Table 10. Here, the reason lies indeed in the data—"VW" refers not only to a German car company, but it is also the initials of a wide variety of people (some apparently involved in religious activities).

Here is a more subtle example of ambiguity: most people would reject the hypothesis that the answer *liver* has the type *location* (as does our redundancy-based type checker). Nevertheless, most people would accept *liver* as an answer to the question *Where is a hepatitis located?*. The problem here is that the EAT `location#n#1` is ambiguous. It is an open question what level of generality of answer types is most appropriate. For instance, it is an error for `RBTC_ig` to reject *Mount Everest* as the answer to a location question because it judges it more likely to be type `thing#n#12`.

But is it not just the EATs that can have more than one reading: answer types in the comparison set can also be ambiguous. Take, for example, the candidate answer *green*, which is rejected as the answer to a color question because it is considered to be more likely to be of the type `light#n#7`. This is the result of frequent occurrences of *light green*, in which the word "light" has a different sense than `light#n#7`.

[11] To improve the readability of this section we will sometimes use simple names for the answer types rather than the correct synset.

[12] There is extensive discussion of problems with Google hit counts in the Language Log blog, e.g., in the post http://itre.cis.upenn.edu/~myl/languagelog/archives/001837.html and the posts linked to from there.

7.2.2. False positives: accepting an incorrectly typed answer

Studying the wrongly accepted incorrectly typed answers is more difficult, as we have to guess for such an answer why none of the answer types in the comparison sets is a more likely type than the EAT of the question. There are two main observations: we have poor comparison sets, and there are again data-related issues that make type checking difficult. One interesting and unexpected problem is related to errors in the answer type extraction.

Issues related to answer type extraction. The overall accuracy of our expected answer type extraction method lies at around 85%, as mentioned in Section 4.1. This means that we can expect around 15% of our questions to be assigned an incorrect EAT. One example where this goes wrong is TREC question 1395 *Who is Tom Cruise married to?* where type `cruise#n#1` is assigned. It is clear that the (at the time) correct answer *Nicole Kidman* is not of this type, nor are any of the incorrect candidate answers. But instead of filtering out *all* answers, no answer is actually rejected, because there is no type in the comparison set which is better correlated to the candidate answers than `cruise#n#1`. In fact, no semantic filtering happens whatsoever because given this incorrect EAT, the types in the comparison set are all as unrelated to the answer as the erroneous EAT itself.

Poor comparison sets. This leads us to the issue of the choice of comparison sets, which is the most sensitive element of our proposed method. Choosing siblings of the EAT that are not named entities and are not leaves in the WordNet hierarchy provides us with a generic method, which has its limits. The most trivial failure happens when the comparison set is actually empty. We see this even in one of the eight answer types we chose for manual evaluation, `continent#n#1` (actually the answer type with the most answers in our test set). Overall, this happens in 29 out of 159 cases.

But even if the comparison set is not empty, it often does not contain sufficient distinct answer types to discern between correctly typed and incorrectly typed answers. The quantitative results presented in Table 8 show a link between the size of the comparison sets and the acceptance of incorrect answers. Answer types with insignificant comparison sets usually have a high acceptance rate of incorrect answers, such as `president#n#3` or `river#n#1` (compare Table 3). Unfortunately, with the decrease of acceptance of incorrect answers also comes the increase of rejected correct answers, which means that finding the right balance for a comparison set is crucial.

Data related issues. Finally, some incorrectly typed answers fail to be rejected because of the data itself, as for example, when there is a strong syntactic link between a particular class of answers with an EAT, although the EAT is not the semantic type of the answer. One example is the link between the terminology of basketball and of colleges. Because so many professional basketball players used to play for college teams, and because many college teams have the same names as professional teams, the scores of professional basketball teams or even players with answer type `college#n#1` is extremely high. Therefore, a number of answers related to professional basketball are accepted as answers for questions about colleges.

This problem, though clearly inherent to our data-driven methods, also touches on the issue of the comparison set, because there is no alternative answer type in the comparison set of `college#n#1` which could possibly discern answers of a basketball-related type from a college-related type. Possibly, this is an example where a generic approach such as redundancy-based type checking is bound to fail.

7.2.3. Lessons learned

Quantitative and qualitative studies did not provide clear-cut answers to the question of whether any of the measures should be preferred over the others, although we try to address this question below. What we do see from our analyses, though, is that the composition of the comparison sets is a significant factor in the performance of our type checking method.

In order to get a better sense of the impact of the choice of comparison sets on our type checking method, we ran additional experiments in which we extended the small comparison sets for the three types *country*, *continent* and *river* by hand and ran the two methods `IG` and `STO` again. For the EAT *country* we added the four types *province*, *river*, *ocean* and *continent*; for the EAT *continent*, we added *province*, *river* and *ocean*; and for the EAT *river*, we added *ditch*, *canal*, *rapid* and *sea*. For the two answer types with small comparison sets, *river* and *continent*, the results changed dramatically, the rate of incorrectly rejected answers increasing as much as the percentage of correctly rejected answers. For type *country*, which had a relatively exhaustive comparison set, there were much less dramatic changes.

This small experiment shows that, even "by hand", it is not straightforward to construct good comparison sets which allow us to discern between correctly and incorrectly typed answers. It also shows that the generic types are

not so bad in practice. Nevertheless, the choice of comparison sets is definitely an issue for future work. With respect to this choice, two research questions might be relevant: is a comparison set better suited to discerning correctly from incorrectly typed answers if it actually contains the real semantic types of the candidate answers? Or would we be better off simply taking a random sample of WORDNET synsets as comparison set? Additionally, we could consider learning ideal comparison sets from our test data automatically.

When should you be using a type checker, and if so which? The qualitative analysis of this subsection touches only very lightly on the differences between the scoring methods we employ in RBTC, and thus, given the indirect way in which the scores are deployed in the RBTC algorithm, it is hard to guess at the reasons for the variation in performance. We will not attempt to do so here. Instead, taking into consideration the tentative conclusions we draw in light of our manual evaluation—that the acceptance rate of correct answers might be a reasonable estimate of recall and that increased rejection of incorrect answers might be extrapolated to increased precision—we simply note that the different scoring methods result in different behaviors which may be useful in different settings.

For instance, the measures based on explicit type statements `STO` and, especially, `LTO` achieve high acceptance rates of correct answers while maintaining relatively good rejection of incorrect answers. In a setting in which it is important to have access to as many possibly correct answers as possible, e.g., in a setting where patents are being accessed, one of these measures, with their presumably high recall, might be suitable. Alternatively, in a setting where wrong answers are more of a problem than incompleteness, e.g., in the medical domain, the information gain measure `IG`, which, under our assumptions, achieves the best precision, would be appropriate.

7.3. A word on performance

An important point that was not discussed so far is the practical applicability of redundancy-based methods, which have shown to be effective, but are still extremely inefficient. The experiments using the RBTC techniques that we describe above took weeks to perform, whereas it takes just minutes to type check the same set of questions with knowledge intensive methods. The mathematics behind this inefficiency are simple: given 25.788 candidate answers and an average size of 5 types per comparison set, we need more than a million queries for all the measures.

There are several solutions. First, to reduce the number of queries, we can reduce the size of comparison sets, and we can focus on the "best" measures in future research. But this will not be enough to allow for an on-line application of the techniques. The more interesting solution is to use a local corpus, which would decrease access time to the hit-counts statistics significantly.

8. Conclusion

Many open domain QA systems answer questions by first extracting a huge number of candidate answers from a document collection, and then picking the most promising one from the list. One criterion for this answer selection is whether the candidate answer is of the semantic type which is expected by the question. In the absence of knowledge sources that provide typing information, the redundancy of the web may be exploited to obtain information the most likely semantic type of a candidate answer.

We defined a general strategy for building redundancy-based type checkers, built around the notions of *comparison set* and *scoring method*. The former provide a set of types which are related to but sufficiently distinct from an expected answer type for questions to discern correctly typed from incorrectly typed answers. Scoring methods are meant to capture the relation between a candidate answer and an answer type. A large part of the paper was devoted to different scoring methods, and we experimented with three types, based on explicit type statements, on co-occurrence statistics for candidate answers and potential types, and on prediction of types from answers where we estimate the dependence of occurrences of types on the occurrence of candidate answers.

We carried out experimental evaluations in two different ways: system-dependent (by looking at the impact of the type checking methods on the overall performance of a particular QA system) and system-independent (by looking specifically at the filtering performance on both correct and incorrect answers gathered from multiple systems). The experiments were complemented with an extensive discussion and error analysis. Our experiments showed that answer type checking can be performed in the absence of rich knowledge sources, and, moreover, that knowledge-poor type checking methods can achieve the same overall performance as knowledge-intensive type checking. Our experiments

did not reveal a clear-cut answer to the question of whether any of the scoring methods should be preferred over the others. Different scoring methods result in different behaviors which may be useful in different settings.

While much of our efforts went into a detailed study of scoring methods, our error analysis suggests that comparison sets are an aspect of our redundancy-based type checking method where we stand to gain from further research.

One final point. In comparing the results of the best performing redundancy-based measure (lenient type statement occurrence, LTO) to the knowledge-intensive approach, we see that the explicit type statements used by this measure are a closer approximation to engineered knowledge than the relatively coarse measures of co-occurrence in a document. But what is perhaps surprising is that the amount of knowledge available to us on the web is large enough now that such simple web mining efforts as involved in calculating LTO yield results comparable to engineered knowledge, at least within the context of a particular application. This suggests that one of the more promising ways of extending the work in this paper is to extend these kinds of focused mining efforts.

Acknowledgements

We are grateful to our referees for the valuable comments.

References

[1] D. Ahn, V. Jijkoun, G. Mishne, K. Müller, M. de Rijke, S. Schlobach, Using Wikipedia at the TREC QA track, in: E.M. Voorhees, L.P. Buckland (Eds.), The Thirteenth Text REtrieval Conference (TREC 2004), 2005.
[2] D. Ahn, V. Jijkoun, K. Müller, M. de Rijke, S. Schlobach, G. Mishne, Making stone soup: evaluating a recall-oriented multi-stream question answering stream for Dutch, in: C. Peters, P.D. Clough, G.J.F. Jones, J. Gonzalo, M. Kluck, B. Magnini (Eds.), Multilingual Information Access for Text, Speech and Images: Results of the Fifth CLEF Evaluation Campaign, 2005.
[3] J.-B. Berthelin, G. De Chalendar, F. Elkateb-Gara, B. Grau O. Ferret, M. Hurault-Plantet, G. Illouz, L. Monceaux, I. Robba, Getting reliable answers by exploiting results from several sources of information, in: R. Bernardi, M. Moortgat (Eds.), Questions and Answers: Theoretical and Applied Perspectives, 2003.
[4] D. Cai, C.J. van Rijsbergen, A case study for automatic query expansion based on divergence, DCS Tech Report TR-2004-172, Department of Computing Science, University of Glasgo, 2004, http://www.dcs.gla.ac.uk/publications/paperdetails.cfm?id=7714.
[5] J. Chu-Carroll, J. Prager, C. Welty, K. Czuba, D. Ferrucci, A multi-strategy and multi-source approach to question answering, in: Proceedings of TREC 2002, 2003.
[6] K. Church, P. Hanks, D. Hindle, W. Gale, Using statistics in lexical analysis, in: U. Zernik (Ed.), Lexical Acquisition: Using Online Resources to Build a Lexicon, Lawrence Erlbaum, 1991, pp. 115–164.
[7] C.L.A. Clarke, G.V. Cormack, G. Kemkes, M. Laszlo, T.R. Lynam, E.L. Terra, P.L. Tilker, Statistical selection of exact answers, in: Proceedings of TREC 2002, 2003.
[8] H. Cui, K. Li, R. Sun, T.-S. Chua, M.-Y. Kan, National University of Singapore at the TREC 13 question answering main task, in: E.M. Voorhees, L.P. Buckland (Eds.), The Thirteenth Text REtrieval Conference (TREC 2004), 2005.
[9] W. Daelemans, J. Zavrel, K. van der Sloot, A. van den Bosch, TiMBL: Tilburg Memory Based Learner, version 5.0, Reference Guide, ILK Technical Report 03-10, 2003, http://ilk.kub.nl/downloads/pub/papers/ilk0310.ps.gz.
[10] T. Dalmas, B. Webber, Information fusion for answering factoid questions, in: R. Bernardi, M. Moortgat (Eds.), Questions and Answers: Theoretical and Applied Perspectives, 2003.
[11] G. de Chalendar, T. Dalmas, F. Elkateb-Gara, O. Ferret, B. Grau, M. Hurault-Plantet, G. Illouz, L. Monceaux, I. Robba, A. Vilnat, The question answering system QALC at LIMSI: experiments in using Web and WordNet, in: Proceedings of TREC 2002, 2003.
[12] T. Dunning, Accurate methods for the statistics of surprise and coincidence, Computational Linguistics 19 (1) (1993) 61–74.
[13] Geographic Names Information System, http://geonames.usgs.gov/stategaz/index.html.
[14] GEOnet Names Server, http://gnswww.nga.mil/geonames/GNS/index.jsp.
[15] E. Hovy, H. Hermjakob, M. Junk, C.-Y. Lin, Question answering in Webclopedia, in: Proceedings of TREC-9, 2000.
[16] V. Jijkoun, M. de Rijke, Answer selection in a multi-stream open domain question answering system, in: Proceedings of 26th European Conference on Information Retrieval (ECIR 2004), in: LNCS, Springer, in press.
[17] V. Jijkoun, J. Kamps, G. Mishne, C. Monz, M. de Rijke, S. Schlobach, O. Tsur, The University of Amsterdam at TREC 2003, in: TREC 2003 Notebook Papers, 2003.
[18] J. Lin, B. Katz, Question answering from the web using knowledge annotation and knowledge mining techniques, in: Proceedings of Twelfth International Conference on Information and Knowledge Management (CIKM 2003), 2003.
[19] B. Magnini, M. Negri, R. Prevete, H. Tanev, Is it the right answer? Exploiting web redundancy for answer validation, in: Proceedings of ACL 40th Anniversary Meeting (ACL-02), University of Pennsylvania, Philadelphia, PA, 2002, pp. 425–432.
[20] C. Manning, H. Schütze, Foundations of Statistical Natural Language Processing, MIT Press, Cambridge, MA, 1999.
[21] G.A. Miller, WordNet: A lexical database, Comm. ACM 38 (11) (1995) 39–41.
[22] D. Moldovan, S. Harabagiu, M. Paşca, R. Mihalcea, R. Girju, R. Goodrum, V. Rus, The structure and performance of an open domain question answering system, in: Proceedings of ACL 2000, 2000, pp. 563–570.

[23] D. Moldovan, M. Paşca, S. Harabagiu, M. Surdeanu, Performance issues and error analysis in an open-domain question answering system, ACM Trans. Inform. Syst. 21 (2003) 133–154.
[24] C. Monz, M. de Rijke, Tequesta: The University of Amsterdam's textual question answering system, in: Proceedings of TREC 2001, 2002, pp. 519–528.
[25] J. Prager, E. Brown, A. Coden, D. Radev, Question-answering by predictive annotation, in: Proceedings of SIGIR 2000, 2000, pp. 184–191.
[26] J. Prager, J. Chu-Carroll, K. Czuba, C. Welty, A. Ittycheriah, R. Mahindru, IBM's PIQUANT in TREC 2003, in: TREC 2003 Conference Notebook, 2003, pp. 36–45.
[27] Quartz, http://ilps.science.uva.nl/~qa/.
[28] J.R. Quinlan, Induction of decision trees, Machine Learning 1 (1986) 81–106.
[29] Quiz-zone, http://www.quiz-zone.co.uk/.
[30] S. Schlobach, M. Olsthoorn, M. de Rijke, Type checking in open-domain question answering, in: Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), IOS Press, Amsterdam, 2004, pp. 398–402.
[31] Text REtrieval Conference (TREC), http://trec.nist.gov.
[32] TreeTagger, http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/.
[33] E.M. Voorhees, Overview of the TREC 2003 question answering track, in: The Twelfth Text REtrieval Conference (TREC 2003), 2004, pp. 54–68.
[34] E.M. Voorhees, D.K. Harman, Appendix: Common evaluation measures, in: Proceedings of TREC 2002, 2003.
[35] J. Xu, A. Licuanan, J. May, S. Miller, R. Weischedel, TREC 2002 QA at BBN: Answer selection and confidence estimation, in: Proceedings of TREC 2002, 2003.