Web-Scale Extension of RDF Knowledge Bases from Templated Websites

Lorenz Bühmann*[1], Ricardo Usbeck*[12], Axel-Cyrille Ngonga Ngomo[1], Muhammad Saleem[1], Andreas Both[2], Valter Crescenzi[3], Paolo Merialdo[3], and Disheng Qiu[3]

[1] Universität Leipzig, IFI/AKSW {lastname}@informatik.uni-leipzig.de
[2] Unister GmbH, Leipzig {firstname.lastname}@unister.de
[3] Università Roma Tre {lastname}@dia.uniroma3.it

Abstract. Only a small fraction of the information on the Web is represented as Linked Data. This lack of coverage is partly due to the paradigms followed so far to extract Linked Data. While converting structured data to RDF is well supported by tools, most approaches to extract RDF from semi-structured data rely on extraction methods based on ad-hoc solutions. In this paper, we present a holistic and open-source framework for the extraction of RDF from templated websites. We discuss the architecture of the framework and the initial implementation of each of its components. In particular, we present a novel wrapper induction technique that does not require any human supervision to detect wrappers for web sites. Our framework also includes a consistency layer with which the data extracted by the wrappers can be checked for logical consistency. We evaluate the initial version of REX on three different datasets. Our results clearly show the potential of using templated Web pages to extend the Linked Data Cloud. Moreover, our results indicate the weaknesses of our current implementations and how they can be extended.

1 Introduction

The Linked Open Data (LOD) Cloud has grown from 12 datasets (also called knowledge bases) to over 2000 knowledge bases in less than 10 years.[4] This steady growth of the LOD Cloud promises to continue as very large datasets such as Linked TCGA [20] with 20.4 billion triples are added to it. However, the LOD Cloud still contains only a fraction of the knowledge available on the Web [13]. This lack of coverage is mainly due to the way the data available on the LOD Cloud is extracted. Most commonly, the data in the LOD Cloud originates from one of two types of sources: structured data (especially databases such as Drugbank,[5] Diseasome,[6] etc.) and semi-structured data sources (for example data extracted from the Wikipedia[7] infoboxes). While generating RDF triples from structured data (especially databases) is well supported by tools such as Triplify [3], D2R [4] and SPARQLMap [21] , devising automatic means to generate

* Both authors contributed equally to this work.
[4] `http://stats.lod2.eu/`
[5] `http://www.drugbank.ca`
[6] `http://diseasome.eu`
[7] `http://wikipedia.org`

RDF from semi-structured data is a more challenging problem. Currently, this challenge is addressed by ad-hoc or manual (e.g., community-driven) solutions. For example, the well-known DBpedia [2] provides a mapping Wiki[8] where users can explicate how the content of infoboxes is to be transformed into RDF. On the one hand, manual approaches offer the advantage of leading to high-precision data; on the other hand, they suffer of a limited recall because of the small number of web sources from which the data is extracted. For example, DBpedia only contains a fraction of the movies that were published over the last years because it was extracted exclusively from Wikipedia. Moreover, the same knowledge base only contains a fraction of the cast of some of the movies it describes.

The main aim of this paper is to address the challenge of extracting RDF from semi-structured data. We introduce REX, an open-source framework for the extraction of RDF from templated websites (e.g., Wikipedia, IMDB, ESPN, etc.). REX addresses the extraction of RDF from templated websites by providing a modular and extensible architecture for learning XPath wrappers and extracting consistent RDF data from these web pages. Our framework is thus complementary to RDF extraction frameworks for structured and unstructured data. While REX targets the extraction of RDF from templated websites in its current version, the architecture of the framework is generic and allows for creating versions of the tool that can extract RDF from other sources on websites, for example from unstructured data or from the billions of tables available on the Web. Our framework has the following features:

1. *Extensibility*, i.e., our framework is open-source, available under the MIT license and can thus be extended and used by any third party;
2. *Use of standards*, i.e., REX relies internally on widely used libraries and on W3C Standards such as RDF, SPARQL and OWL;
3. *Modularity*, i.e., each of the modules can be replaced by another implementation;
4. *Scalability*, i.e., the current algorithms can be used on large amounts of data;
5. *Low costs*, i.e., REX requires no human supervision;
6. *Accuracy*, i.e., the current implementation achieves satisfactory F-measures and
7. *Consistency*, i.e., REX implements means to generate triples which abide by the ontology of the source knowledge base providing the training data.

In addition to being novel in itself, REX introduces a *novel wrapper induction technique* for extracting structured data from templated Web sites. This induction approach makes use of the large amount of data available in the LOD Cloud as training data. By these means, REX circumvents the problem of high annotation costs faced by several of the previous wrapper induction approaches [16, 11] while keeping the high accuracy of supervised wrapper induction methods. By post-processing the output of website wrappers, our system can generate novel triples. To ensure that these novel triples are consistent, REX provides a consistency check module which computes and uses the axioms which underlie the input knowledge base K. Only those triples which do not break the consistency rules are returned by REX. The contributions of this paper are consequently as follows:

[8] `http://mappings.dbpedia.org`

1. We introduce a novel framework for the extraction of RDF from templated websites.
2. We present a novel wrapper induction approach for the extraction of subject-object pairs from the Web.
3. We integrate state-of-the-art disambiguation and schema induction techniques to retrieve high-quality RDF.
4. We evaluate the first version of REX on three datasets and present both the strengths and weaknesses of our approach.
5. Overall, we present the (to the best of our knowledge) first web-scale, low-cost, accurate and consistent framework that allows extracting RDF from structured websites.

The rest of this paper is organized as follows. In Section 2, we introduce the notation that underlies this paper and the problems that we tackle. Section 3 presents the architecture of REX in more detail as well as the current implementation of each of its components. In particular, we illustrate our approach to generate examples from a knowledge base K and we show our algorithm to learn web wrappers from such examples. Subsequently, we give an overview of AGDISTIS [22] which we use to address the problem of URI disambiguation. Finally, we describe our current solution to ensuring the validity of the data generated by REX. In Section 4 we present the results of REX on 3 datasets, each containing at least 10,000 pages. We discuss related work in Section 5, and we conclude the paper in Section 6. More information on REX can be found at `http://aksw.org/Projects/REX` including inks to the source code repository (incl. examples), to the documentation and to a tutorial of the framework.

2 Notation and Problem Statement

In this section, we present the concepts and notation to understand the concept behind REX. We denote RDF triples as $< s, p, o >$ where $(s, p, o) \in R \times P \times (R \cup L)$. We call R the set of resources, P the set of properties and L the set of literals. We call $A = R \cup P \cup L$ the set of all atoms. We regard knowledge bases K as sets of triples. We denote the set of all pairs (s, o) such that $< s, p, o > \in K$ with $pairs(p, K)$. We define the in-degree $in(a)$ of an atom a in K as the number of distinct x such that there is a predicate q with $< x, q, a > \in K$. Conversely, the out-degree $out(a)$ of a is defined as the number of distinct atoms y that are such that there exists a predicate q' with $< a, q', y > \in K$. We assume the existence of a labeling function $label$, which maps each element of A to a sequence of words from a dictionary D. Formally, $label : A \to 2^D$. For example, the value of $label(\texttt{r})$ can be defined as the set of `x` with `<r, rdfs:label, x>`$\in K$ if `r` is a resource and as the lexical form of `r` if `r` is a literal.

Based on this formalisation, we can define the problem that REX addresses as follows: Given (1) a predicate p that is contained in a knowledge base K and (2) a set of unlabeled web pages $W = \{w_1, w_2, \ldots, w_{|W|}\}$, extract a set of triples $< s_i, p, o_i >$ from the websites of W. Several tasks have to be addressed and solved to achieve this goal within the paradigm that we adopt:

Problem 1: We first require an approach for extracting pairs of resource labels out of unlabelled pages w_i. We tackle this problem by means of a wrapper induction algorithm (see Section 3.4). We assume that we are given (1) a set $E \subseteq \{(s, o) : < s, p, o > \in K\}$ of positive examples for a predicate p from the Linked Data Web and (2) a set of web pages W without any labeling. Our aim is to generate high-quality wrappers, expressed as pairs of XPath expressions over these unlabeled web pages W, that extract a pair of values from each page.

Problem 2: Once the pairs of values have been extracted from web pages, we need to ground them in the knowledge base K. In this context, grounding means that for each value extracted by our solution to Problem 1 we have to either (1) find a matching resource or (2) generate a novel resource or literal for this particular value. We address this challenge by using a URI disambiguation approach that combines breadth-first search and graph algorithms to determine a resource that matches a given string. If no URI is found, our approach generates a new resource URI (see Section 3.5).

Problem 3: Once new knowledge has been generated, it is central to ensure that the knowledge base K to which it is added remains consistent. To this end, we need to ensure that we do not add any statements to K that go against its underlying axioms. The problem here is that these axioms are not always explicated in knowledge bases in the LOD Cloud. We thus devise an approach to generate such axioms from instance data (see Section 3.5). To achieve this goal, we use a statistical analysis of the use of predicates across the knowledge base K. Moreover, we provide means to use RDFS inference to generate new knowledge from new resources generated by our solution to Problem 2.

3 The REX Framework

In the following, we present REX, an integrated solution to the three problems presented above. We begin by giving an overview of its architecture. Then, we present each of its components. As running example, we use the extraction of movie directors from web pages.

3.1 Overview

Figure 1 gives an overview of REX. All modules are interfaces, for which we provide at least one implementation. Hence, REX can be ran out of the box. Given a predicate p and a knowledge base K, REX provides a domain identification interface, which allows for detecting Web domains which pertain to this predicate. For example, the predicate `dbo:actor` leads to the domain `http://imdb.com` being retrieved. From this domain, a set W of web pages can be retrieved by using a *crawler*. The results of the crawling are stored in a solution for unstructured data, for example an index. REX then generates a set of examples using an instantiation of the *example generator* interface. The goal here is to generate a sample E of all elements of $pairs(p, K)$ that allows learning high-quality pairs of XPath expressions. The examples are given to a *wrapper* *inducer*, which learns pairs of XPath expressions for extracting the pairs of values in E from the elements of W. These pairs are then applied to all pages of W. The extraction results, i.e., pairs of strings, are passed on to a *URI generator*, which implements a graph-based disambiguation approach for finding or generating URIs for the strings contained in the extraction results. The resulting set C of candidate triples are finally forwarded to an *validation engine*, which learns axioms from K and applies these to C to derive a set of triples that are consistent with K. In the following, we detail our current implementation of each of these components.

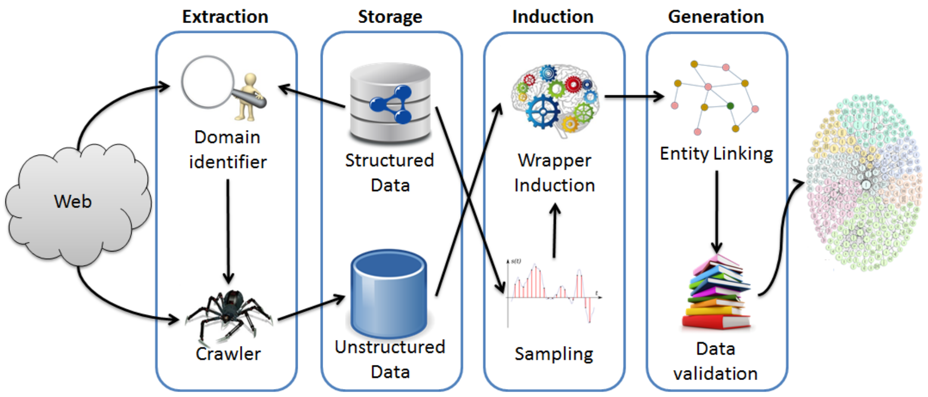


Fig. 1: Architecture of REX.

3.2 Extraction Layer

REX's data extraction layer consists of two main components: The *domain identification module* is the first component of the layer and takes a set of triples (s, p, o) as examples and returns a ranked list of Web domains. Our current implementation simply uses the Google interface to search for websites that contain the label of all s, p and o. The top-10 domains for each triple are selected and their rank is averaged over all triples. The resulting ranking is returned. For our example `dbo:actor`, we get `http://imdb.com` as top-ranked domain. The second component consists of a *crawler interface* which allows to gather the web pages that are part of the detected domain and collect them in a storage solution for unstructured data. Currently, we rely on crawler4j[9] for crawling and Apache Lucene[10] for storing the results of the crawling.

3.3 Storage Layer

The storage layer encapsulates the storage solutions for structured data (i.e., the knowledge base K) and the unstructured data (i.e., the output of the extraction layer). We

[9] `https://code.google.com/p/crawler4j/`
[10] `http://lucene.apache.org/`

assume that the structured data can be access via SPARQL. The unstructured data storage is expected to return data when presented with a pair (s, o) of resources, which is commonly a positive or negative example for the pairs that abide by p. As stated above, we rely on a Lucene index that can access the labels of resources and simply search through its index for pages that contain both a label for s and a label for o.

3.4 Induction Layer

The induction layer uses the data in the storage layer to compute wrappers for the website crawled in the first step. To this end, it contains two types of modules: The *example generation* module implements sampling algorithms that are used to retrieve examples of relevant pairs (s, o) such that $(s, p, o) \in K$. These examples are used to feed the *wrapper induction* module, which learns the wrappers that are finally used to extract data from web pages. Hereafter, we present the implementations of these modules.

Generation of Examples Given a knowledge base K, the generation of all examples E for a predicate p can be retrieved by computing all triples $< s, p, o >$ from K. However, using all triples might lead to poor scalability, especially if K is very large. To ensure the scalability of our approach, we thus aimed to ensure that we can provide REX with only a sample of E and thus reduce its learning runtime without diminishing its accuracy. Our first intuition was that it is more likely to find resources that stand for well-known real-world entities on the Web. Thus, by selecting the most prominent examples from the knowledge K, we should be able to improve the probability of finding web pages that contain both the subject and the object of our examples. This intuition can be regarded as prominence-driven, as it tries to maximize the number of annotated pages used for learning. We implemented this intuition to generating a sample of E by implementing a first version of the example generator that ranks the examples (s, o) in E in descending order by how prominent they are in the knowledge base. The score scr for ranking the examples was computed by summing up the in- and out-degree of s and o: $scr(s, o) = in(s) + in(o) + out(s) + out(o)$. We call this example selection *prominence-based*.

The main drawback of this first intuition is that it introduces a skew in the sampling as we only consider a subset of entities with a particular distribution across the pages in W. For example, actors in IMDB have different templates depending on how popular they are. Learning only from popular actors would then lead to learning how to extract values only from web pages obeying to particular type of HTML template. While this problem can be by choosing a large number of examples, we revised our sampling approach to still use the ranking but to sample evenly across the whole list of ranked resources. To this end, given a number n of required pairs, we return the first n pairs (s, o) from the ranked list computed above whose index idx abides by $idx(s, o) \equiv 0 \left(mod \left\lfloor \frac{|E|}{n} \right\rfloor \right)$. We call this second implementation of the example generator interface the *uniform* approach .

Wrapper Generation Detecting rules to extract the subject-object pairs related to a property p is the most difficult step when aiming to extract RDF from templated website. Here, we present our current implementation of the wrapper induction module interface of REX, which aims to extract subject-object pairs for p from a set of pages W that belong to the same website and share a common template. We assume that an *example generator* provides the input set E containing a subset of the pairs that can be extracted from the pages in W. Formally, let Q denote the set of pages that contain a pair in E: $Q = \{w : w \in W,\ (s, o) \in E \wedge (label(s), label(o)) \in\ w\}$, where $(label(s), label(o)) \in w$ denotes that at least one of the labels of s and at least one of the labels of o occur in the page w. We use the pairs in E to gain the positive annotations for the pages in Q. These annotations are needed to automatically infer a set of wrappers, i.e., a set of extraction rule pairs that extract the target subject-object pairs.

To avoid the extraction of incorrect values, our approach includes a technique to evaluate the output wrapper coverage, i.e., the number of pages in W for which the wrappers inferred from Q correctly extract the target subject-object pairs.

Listing 1 reports the pseudo-code of our algorithm to generate the wrappers that extract subject-object pairs related to a property p from a set of pages: it takes as input the set of pages W and the set of examples E. To abstract the extraction rules generative process in our implementation, we assume that there exists, as a parameter of the algorithm, a class of all the creatable extraction rules $\mathcal{R}$. It corresponds to the set of XPath expressions that we can generate over the pages in W.

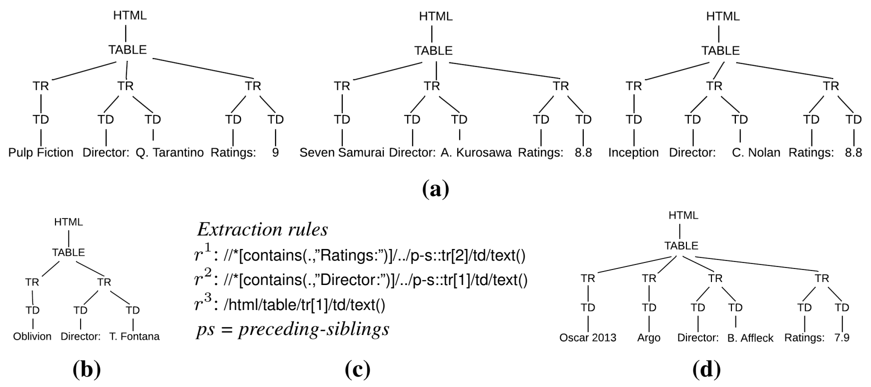
As a first step (line 2), the algorithm computes the set of pages Q (we assume $Q \neq \emptyset$). Then, it picks up a small set of sample pages I from Q.[11] From the pages in I two initial sets of extraction rules, R_s and R_o, are generated (lines 4-5), as follows. First, we analyze the DOM tree of the pages to locate nodes that are part of the template. We use these nodes as roots of XPath expressions that match with the input pair. To discover the template nodes, we compute the occurrences of the textual leaf nodes in the pages. Following the intuition developed in [1], we consider template nodes the document root, the nodes with an *id* attribute, and the text leaves that occur exactly once with same value and same root-to-leaf sequence of tags in a significant percentage (80%) of pages. The rationale is that it is very unlikely that a node occurs exactly once in several pages with the same root-to-leaf path by chance; rather, it is likely repeated in every page since it comes from a piece of the underlying HTML template.

Template nodes are then used as *pivot nodes* to generate XPath expressions that match with nodes containing a textual leaf that equals the subject (object) of the input pair. Given a pivot node l, an XPath expression for the textual node t is computed by appending three expressions: (i) an expression that matches with the pivot node t, (ii) the path from t to the first ancestor node, n_{lt}, shared by t and l, (iii) the path from n_{lt} to l (which descends from the shared ancestor node to the target textual node). To avoid an excessive proliferation of rules, we bound the length of the XPath expressions, i.e., the number of XPath steps.[12]

The above step produces several extraction rules that correctly work on the pages in I. However some of these rules could not work on a larger set of pages. For example,

[11] In our implementation $k = |I| = 10$.

[12] We observed that producing rules longer than 8 steps do not produce any benefit.



Extraction rules
r^1: //*[contains(.,"Ratings:")]/../p-s::tr[2]/td/text()
r^2: //*[contains(.,"Director:")]/../p-s::tr[1]/td/text()
r^3: /html/table/tr[1]/td/text()
ps = preceding-siblings
(c)

Fig. 2: **(a)** DOM trees of three pages (in a fictional set I), **(b)** a page in Q (with a template that differs from those of the pages in I), **(c)** some rules to extract the movie title, and **(d)** a page in W (with a template that differs from those of the pages in Q).

consider a set of pages such as those shown in Figure 2(a). Assuming that the leaf nodes 'Director:' and 'Ratings:' appear once with the same root-to-leaf path in most of the pages in I, they would be considered as template nodes. Figure 2(c) reports an example of the XPath expressions pivoted in these nodes, and generated to extract the movie title. Notice, however, that rule r^1 does not extract the movie title on pages like that depicted in Figure 2(b), i.e., pages without user ratings. To improve the accuracy of the rules generated from pages in I, we evaluate the generated rules over Q, and select those that extract the largest number of annotations (line 6). In our example, the extraction rules r^2 and r^3 would be selected, while r^1 would be discarded, as the former rules work also on the page of Figure 2(b), while the latter does not.

The selected rules are those better working for the pages in Q, that are the pages containing pairs of K. Although it is likely that these rules also work for the whole collection of input pages, it might also be the case that W contains pages obeying to a slightly different template not observed within Q. For example, consider the page in Figure 2(d): since the movie has been awarded 3 Oscars, the corresponding page has small structural differences, and neither r^1 nor r^3 correctly extract the title.

To overcome this issue, we leverage the redundancy of equivalent rules generated in the above steps. Targeting only resources from pages for which the extraction is likely to work correctly, we return the pairs (lines 7-8) on which all the distinct yet equivalent rules return the same value. Again from our example, observe that rules r^2 and r^3 extract different values from the page in Figure 2(d) (*Argo* and *Oscar 2013*, respectively), therefore, none of the values extracted from that page would be added in the final output. All these rules are used later (lines 9-13) to check that they extract the same value (line 10) from a web page.

Listing 1 ALFREX: Extract Subject-Object Pairs from a Website

Input: knowledge base K, a predicate p, a set of examples $E = \{(s, o)|(s, p, o) \in K\}$
Input: a set of pages $W = \{w_1, \ldots, w_{|W|}\}$ containing data related to the predicate p

Parameter: a class of extraction rules $\mathcal{R}$ over W
Parameter: k, the number of sample pages for generating the rules

Output: set T of pairs of strings extracted from pages W

```
1: $T := \emptyset$; // output pairs of strings
2: $Q := \{w \in W\colon (label(s), label(o)) \in w, (s, o) \in E\}$;
3: $I :=$ a set of $k$ random pages from $Q$;
4: $R_s := \{r, r \in \mathcal{R}, w \in I, (label(s), label(o)) \in w, r(w) = label(s)\}$;
5: $R_o := \{r, r \in \mathcal{R}, w \in I, (label(s), label(o)) \in w, r(w) = label(o)\}$;
6: $(r_s, r_o) := \text{argmax}_{r_s \in R_s, r_o \in R_o} |\{w, w \in Q, (label(s), label(o)) \in w, r_s(q) = label(s)$ and $r_o(q) = label(o)\}|$;
7: $\{r_s^1, r_s^2, \ldots, r_s^n\} \leftarrow \{r, r \in R_s, r(Q) = r_s(Q)\}$;
8: $\{r_o^1, r_o^2, \ldots, r_o^m\} \leftarrow \{r, r \in R_o, r(Q) = r_o(Q)\}$;
9: for $q \in W$ do
10:     if $(r_s^1(q) = \ldots = r_s^n(q)$ and $r_o^1(q) = \ldots = r_o^m(q))$ then
11:         $T \leftarrow T \cup \{(r_s^1(q), r_o^1(q))\}$;
12:     end if
13: end for
14: return $T$;
```

3.5 Generation Layer

Now that data has been extracted from the websites, REX is ready to generate RDF out of them. To achieve this goal, two steps needs to be carried out. First, the strings retrieved have to be mapped to RDF resources or literals. This is carried out by the *URI disambiguation* modules. The resulting triples then need to be checked for whether they go against the ontology of the knowledge base or other consistency rules. This functionality is implemented in the *data validation* modules.

URI Disambiguation URI disambiguation is not a trivial task, as several resources can share the same label in a knowledge base. For example, "Brad Pitt" can be mapped to the resource `:Brad_Pitt` (the movie star) or `:Brad_Pitt_(boxer)`, an Australian boxer. We address this problem by using *AGDISTIS*, a framework for URI disambiguation [22]. In our current implementation, we chose to simply integrate the AGDISTIS framework using DBpedia 3.8. We chose this framework because it outperforms the state-of-the-art frameworks AIDA [15] and DBpedia Spotlight [18] by 20% w.r.t. its accuracy. Especially on short RSS feeds containing only two resource labels, the approach achieves 3% to 11% higher accuracy. More details on AGDISTIS as well as a thorough evaluation against popular frameworks such as DBpedia Spotlight and AIDA can be found in [22]. Note that if no resources in K has a URI which matches s or o, we generate a new cool URI[13] for this string.

[13] `http://www.w3.org/TR/cooluris`

Data Validation Sequentially applying the steps before results in a set of triples $< s, p, o >$ that might not be contained in K. As we assume that we start from a consistent knowledge base K and the whole triple generation process until here is carried out automatically, we need to ensure that K remains consistent after adding $< s, p, o >$ to K. To this end, REX provides a data validation interface whose first implementation was based on the DL-Learner.[14] Depending on the size of K, using a standard OWL reasoner for consistency checks can be intractable. Thus, our current implementation applies the following set of rules based on the schema of K and add a triple $< s_1, p, o_1 >$ only if it holds that:

1. If a class C is the domain of p, there exists no type D of s_1 such that C and D are disjoint.
2. If a class C is the range of p, there exists no type D of o_1 such that C and D are disjoint.
3. If p is declared to be functional, there exists no triple $< s_1, p, o_2 >$ in K such that $o_1 \neq o_2$.
4. If p is declared to be inverse functional, there exists no triple $< s_2, p, o_1 >$ in K such that $s_1 \neq s_2$.
5. If p is declared to be asymmetric, there exists no triple $< o_1, p, s_1 >$ in K.
6. If p is declared to be irreflexive, it holds that $s_1 \neq o_1$.

Note that this approach is sound but of course incomplete. Although an increasing number of RDF knowledge bases are published, many of those consist primarily of instance data and lack sophisticated schemata. To support the application of the above defined rules, we follow the work in [6, 7], which provides a lightweight and efficient schema creation approach that scales to large knowledge bases.

4 Evaluation

The goal of the evaluation was to provide a detailed study of the behavior of the current REX modules with the aim of (1) ensuring that our framework can be used even in its current version and (2) detecting current weaknesses of our framework to trigger future developments. In the following, we begin by presenting the data and hardware we used for our experiments. Thereafter, we present and discuss the results of our experiments. Detailed results can be found at the project website.

4.1 Experimental Setup

We generated our experimental data by crawling three websites, i.e.,

1. `imdb.com` where we extracted `dbo:starring`, `dbo:starring`$^{-1}$ and `dbo:director`;
2. `goodreads.com`, from which we extracted `dbo:author` and `dbo:author`$^{-1}$;
3. `espnfc.com` with the target relations `dbo:team` and `dbo:team`$^{-1}$.

[14] `http://dl-learner.org`

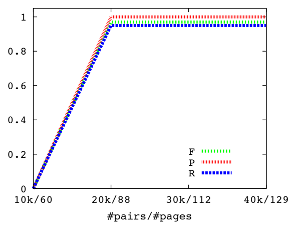
We chose these websites because they represent three different categories of templated websites. `imdb.com` widely follows a uniform template for all pages in the same subdomain. Thus, we expected the wrapper learning to work well here. `goodreads.com` represents an average case of templated websites. While template are most widely used and followed, missing values and misused fields are more common here than in our first dataset. The third dataset, `espnfc.com`, was chosen as worst-case scenario. The dataset contains several blank pages, a large variety of templates used in manifold different fashions. Consequently, defining a set of golden XPaths is a tedious task, even for trained experts. Thus, we expected the results on this dataset to be typical for the worst-case behavior of our approach. We randomly sampled 10,000 HTML pages per subdomain for our experiments and manually built reference XPath expressions to evaluate the precision and recall of the generated extraction rules. The precision, recall and F-measure reported below were computed by comparing the output of REX with the output of the reference XPath expressions. All extraction runtime experiments were carried out on single nodes of an Amazon EC2.small instance.

4.2 Results

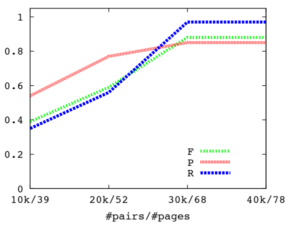
Effect of Number of Examples and Sampling Strategy on F-measure The results of our experiments on altering the number of examples used for learning are shown in Figures 3a-3h. Due to space limitations, we show the average results over all the pairs extraction by our wrapper induction approach for each of the domains. The results achieved using the prominence-based sampling show the expected trend: on pages that use a consistent template (such as the director pages in `imdb.com`), our approach requires as few as around 70 pages for $|Q|$. Once this value is reached, REX can compute high-quality extraction rules and achieves an F-measure of 0.97 (see Figures 3a). For pages that change template based on the prominence of the entities they describe (like the actors' pages, see Figure 3b), our approach requires more training data to achieve a high F-measure. The increase of F-measure is clearly due to an increase in precision, pointing to REX being able to better choose across different alternative XPaths when provided with more information. The results of `goodreads.com` support our conjecture. With more training data, we get an increase in precision to up to 1 while the recall drops, leading to an overall F-measure of 0.89 for 40k examples. In our worst-case scenario, we achieve an overall F-measure close to 0.6. The lower value is clearly due to the inconsistent use of templates across the different pages in the subdomains.

Table 1: Average evaluation results using all available pairs as training data.

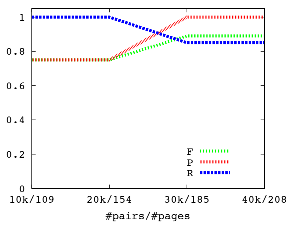
	P	R	F-measure	# pages
`dbo:director`	0.82	1.00	0.89	216
`dbo:starring`	0.86	1.00	0.90	316
`dbo:author`	0.94	0.85	0.86	217
`dbo:team`	0.32	0.43	0.35	656



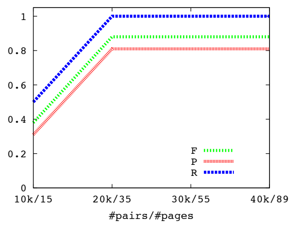
(a) Directors from IMDB, prominence-based sampling



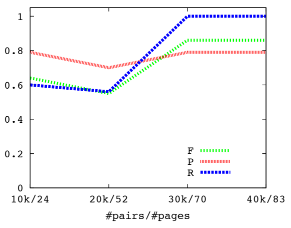
(b) Actors from IMDB, prominence-based sampling



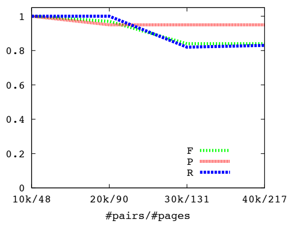
(c) Authors from Goodreads, prominence-based sampling



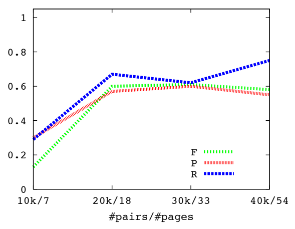
(d) Directors from IMDB, uniform sampling



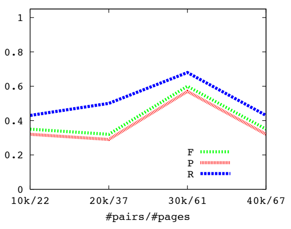
(e) Actors from IMDB, uniform sampling



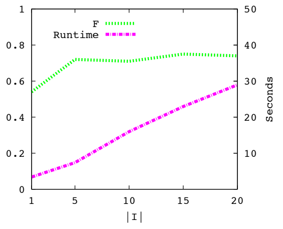
(f) Authors from Goodreads, uniform sampling



(g) Teams from ESPNFC, prominence-based sampling



(h) Teams from ESPNFC, uniform sampling



(i) Average computational time and F-measure over all datasets

Fig. 3: Overall evaluation results of the extraction of pairs. Figures (a)-(h) show the average precision, recall and F-measure achieved the generated XPaths for the prominence-based and uniform sampling. The x-axis shows the number of examples and the number of sample pages retrieved in the format $|E|/|Q|$. Figure (i) shows the average computational time and the corresponding F-measures for different sizes of $|I|$.

The results based on the uniform sampling strategy reveal another trait of REX. As expected, the coverage achieved using uniform sampling is clearly smaller in all cases. The results achieved with all the training data available clearly show the importance of sampling (see Table 1). While one could conjecture that using all data for training would be beneficial for our approach, the F-measures achieved by using all the data suggest that sampling can be beneficial for the extraction, especially when the web pages do not follow a rigid template (e.g., in `esnpfc.com`) or when the data in the knowledge base is noisy. Overall, our results suggest that our approach is accurate, also for pages where

Property	#Possible triples	#Triples generated by AlfREX	#Consistent triples	#Correct triples	#New triples
`dbo:author`$^{-1}$	54	32	32	22	22
`dbo:author`	83	83	69	54	54
`dbo:team`$^{-1}$	2	1	1	0	0
`dbo:team`	30	55	42	19	13
`dbo:starring`$^{-1}$	40	99	83	35	34
`dbo:starring`	70	70	44	33	32
`dbo:director`	61	56	52	41	41

Table 2: Triples generated by 100 randomly sampled pages, number of possible triples generated by using gold standard rules

entities with different prominence are assigned variable templates as in `imdb.com` actors. If multiple occurrences of the same value are present in the same page (as in the case of books, actors and directors), our algorithm is able to detect the most stable one. Moreover, our approach seems robust against noisy labels, even when there are many false positive in the page (e.g., book author pages that include many links to different books by the same author). An important feature of our approach is that it can obtain accurate XPaths even by learning from a very small fraction of pages. For example, in our experiments on up to 40k pages, our approach learned XPath expressions from only 0.5% to 1.16% of $|W|$. Still, for very noisy domains with an inconsistent use of templates, our approach can lead to less accurate extraction rules.

Runtime Performance We evaluated the runtime performance of our approach by using 40k examples and the prominence-based distribution while altering the size of I. As expected, setting $|I|$ to a low value (e.g., 1) leads to less rules being generated and thus to an overall better runtime performance (see Figure 3i). By setting I to a low value, REX can be used to get a quick overview of possible extraction results, a characteristic of our system that could result beneficial for end users. Yet, it also leads to worse overall F-measures. Setting I to a higher value (e.g., 15) leads to a more thorough (i.e., more time-demanding) analysis of the websites and thus to better results. Still overall, our approach scales quasi linearly and requires on average less than thirty seconds to learn wrappers out of existing data even for $|I| = 20$.

Quality of RDF Output To check the quality of the RDF we generated, we manually checked the triples extracted from each property of our three domains. Each triple was checked by at least two annotators, which reached a significant Kohen's kappa score of 0.88 overall. On `goodreads.com` we achieved a precision of 75.24%. While we achieve a precision of 78.85% when extraction directors from *imdb.com* and of 75% on `starring`, the extraction of `starring`$^{-1}$ proves more tedious (precision = 42.17%). As expected, the data extracted from *espnfc.com* has a low precision of 44.19%. The results on `starring`$^{-1}$ are due to the fact that several actors can star in a movie while assuming other roles. Thus, our extraction framework often overgenerates triples and produces false positives (e.g., directors are often included). The results on *espnfc.com* are clearly due to the templates not being used correctly. Still, our results clearly show the potential of our approach, as 60.68% of the triples we extracted are both correct and novel.

5 Related Work

To the best of our knowledge, no open-source framework covers the complete functionality of the REX framework. REX relies internally on URI disambiguation and data validation based on automatically extracted axioms [6]. These are both areas of research with a wide of body of publications. Especially, several approaches to URI disambiguation based on graphs [15, 22] and statistical information from text [18] have been developed recently. The extraction of axioms from knowledge based using statistical information [6, 7] as also flourished over the last years. The main idea underlying these approaches is to use instance knowledge from knowledge bases without expressive schemas to compute the axioms which underlie the said knowledge bases. We refer the reader to the publications above for an overview of these two research areas.

REX is mainly related to wrapper induction. Early approaches to learning web wrappers were mostly supervised (see, e.g., [16, 11]). These systems were provided with annotated pages out of which they infer extraction rules that allow extracting data from other unlabeled pages with the same structure as the annotated pages). For example [16] presents *Tresher*, a system that allows non-technical end-users to teach their browser how to extract data from the Web. Supervised approaches were yet deemed costly due to the human labor necessary to annotate the input web pages. Unsupervised wrapper induction methods have thus been explored [8, 1] to reduce the annotation costs. However, the absence of a supervision often lead these systems to produce wrappers of accuracy not suitable for production level usage. Novel approaches thus aim to minimize the annotation costs while keeping a high precision. For example, the approach presented in [10] relies on the availability of a knowledge base in the form of dictionaries and regular expressions to automatically obtain training data. Recently, [9] describes a supervised framework that is able to profit from crowd-provided training data. The learning algorithm controls the cost of the crowd sourcing campaign w.r.t. quality of the output wrapper. However, these novel approaches do not target the generated of RDF data.

Linked Data has been used to learn wrappers to extract RDF from the Web in recent years. For example, [12] exploits Linked Data as a training data to find instances of given classes such as universities and extract the attributes of these instances while relying on the supervised wrapper induction approach presented in [14]. However, they require a manual exploration of the Linked Data sources to generate their training data, which leads to a considerable amount of manual effort. The DEIMOS project [19] is similar to REX, as it aims at bringing to the Semantic Web the data that are published through the rest of the Web. However, it focuses on the pages behind web forms. OntoSyphon [17] operates in an "ontology-driven" manner: taking any ontology as input, OntoSyphon uses the ontology to specify web searches that identify possible semantic instances, relations, and taxonomic information, in an unsupervised manner. However, the approach makes use of extraction patterns that work for textual documents rather than structured web pages. To the best of our knowledge, none of the existing approaches covers all steps that are required to extract consistent RDF from the Web. Especially, only [19] is able to generate RDF but does not check it for consistency. In contrast, REX is the first approach that is scalable, low-cost, accurate and can generate consistent RDF.

6 Conclusions

In this paper we presented the first framework for the consistent extraction of RDF from templates Web pages. REX is available as open source[15] Java implementation in an easily extendable fashion. Our framework uses the LOD Cloud as source for training data that are used to learn web wrappers. The output of these wrappers is used to generate RDF by the means of a URI disambiguation step as well as a data validation step. We studied several sampling strategies and how they affect the F-measure achieved. Our overall results show that although we can extract subject-object pairs with a high accuracy from well-templated websites, a lot of work still needs to be done in the area of grounding these strings into an existing ontology. One solution to this problem might be to use more context information during the disambiguation step. Moreover, more sophisticated approaches can be used for crawling websites offering structured navigation paths towards target pages [5]. By these means, we should be able to eradicate some of the sources of error in our extraction process. Our approach can be further improved by combining it with crowdsourcing-based approaches for wrapper induction such as ALFRED [9] or by learning more expressive wrappers. We thus regard this framework as a basis for populating the Web of Data using Web pages by professional end-users.

Acknowledgments. Parts of this work were supported by the ESF and the Free State of Saxony. We thank M. Speicher & M. Röder.

References

1. A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD*, pages 337–348, 2003.
2. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, pages 722–735, 2007.
3. S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: light-weight linked data publication from relational databases. In *WWW*, pages 621–630, 2009.
4. C. Bizer and A. Seaborne. D2rq - treating non-rdf databases as virtual rdf graphs. In *ISWC2004 (posters)*, November 2004.
5. L. Blanco, V. Crescenzi, and P. Merialdo. Efficiently locating collections of web pages to wrap. In J. Cordeiro, V. Pedrosa, B. Encarnação, and J. Filipe, editors, *WEBIST*, pages 247–254. INSTICC Press, 2005.
6. L. Bühmann and J. Lehmann. Universal OWL axiom enrichment for large knowledge bases. In *Proceedings of EKAW 2012*, pages 57–71. Springer, 2012.

[15] `http://rex.aksw.org`

7. L. Bühmann and J. Lehmann. Pattern based knowledge base enrichment. In *12th International Semantic Web Conference, 21-25 October 2013, Sydney, Australia*, 2013.
8. V. Crescenzi and P. Merialdo. Wrapper inference for ambiguous web pages. *Applied Artificial Intelligence*, 22(1&2):21–52, 2008.
9. V. Crescenzi, P. Merialdo, and D. Qiu. A framework for learning web wrappers from the crowd. In *Proceedings of the 22nd international conference on World Wide Web*, WWW '13, pages 261–272, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.
10. N. Dalvi, R. Kumar, and M. Soliman. Automatic wrappers for large scale web extraction. *Proc. VLDB Endow.*, 4(4):219–230, Jan. 2011.
11. S. Flesca, G. Manco, E. Masciari, E. Rende, and A. Tagarelli. Web wrapper induction: a brief survey. *AI Communications*, 17(2):57–61, 2004.
12. A. L. Gentile, Z. Zhang, I. Augenstein, and F. Ciravegna. Unsupervised wrapper induction using linked data. In *Proceedings of the seventh international conference on Knowledge capture*, K-CAP '13, pages 41–48, New York, NY, USA, 2013. ACM.
13. D. Gerber, S. Hellmann, L. Bühmann, T. Soru, A.-C. Ngonga Ngomo, and R. Usbeck. Real-time RDF extraction from unstructured data streams. In *Proceedings of the international semantic web conference*, ISWC, 2013.
14. Q. Hao, R. Cai, Y. Pang, and L. Zhang. From one tree to a forest: a unified solution for structured web data extraction. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, SIGIR '11, pages 775–784, New York, NY, USA, 2011. ACM.
15. J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum. Robust Disambiguation of Named Entities in Text. In *Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, Edinburgh, Scotland*, pages 782–792, 2011.
16. A. Hogue and D. Karger. Thresher: automating the unwrapping of semantic content from the world wide web. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 86–95, New York, NY, USA, 2005. ACM.
17. L. McDowell and M. J. Cafarella. Ontology-driven, unsupervised instance population. *J. Web Sem.*, 6(3):218–236, 2008.
18. P. N. Mendes, M. Jakob, A. Garcia-Silva, and C. Bizer. Dbpedia spotlight: Shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems (I-Semantics)*, 2011.
19. R. Parundekar, C. A. Knoblock, and J. L. Ambite. Linking the deep web to the linked dataweb. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*. AAAI, 2010.
20. M. Saleem, S. S. Padmanabhuni, A.-C. Ngonga Ngomo, J. S. Almeida, S. Decker, and H. F. Deus. Linked cancer genome atlas database. In *Proceedings of I-Semantics*, 2013.
21. J. Unbehauen, C. Stadler, and S. Auer. Accessing relational data on the web with sparqlmap. In *JIST*, pages 65–80, 2012.
22. R. Usbeck, A.-C. Ngonga Ngomo, R. Michael, S. Auer, D. Gerber, and A. Both. Agdistis - agnostic disambiguation of named entities using linked open data. In *International Semantic Web Conference*. 2014.