Query Evaluation with Asymmetric Web Services

Nicoleta Preda
Universite de Versailles SQ, Laboratoire PRiSM
Versailles, France

Fabian Suchanek
INRIA Saclay
Paris, France

Wenjun Yuan, University of Hong Kong,
Hong Kong, China

Gerhard Weikum
Max Planck Institute for Informatics,
Saarbruecken, Germany;

March 16, 2011

1 Introduction

Motivation. Recent projects such as DBpedia [3], YAGO [28, 18], freebase.com, wolframalpha.com, trueknowledge.com, KnowItAll [4], and others have successfully constructed semantic knowledge bases of large scale. Factual knowledge is typically represented in RDF, the W3C standard for Semantic-Web contents. RDF data can be seen as a graph whose nodes are entities (e.g., persons, companies, movies, locations) and whose edges are relationships (e.g., bornOnDate, isCEOof, actedIn). These knowledge bases can be queried using the W3C-endorsed SPARQL [33] language.

Yet, a knowledge base about entities can never be fully complete or always up to date. With the ANGIE system [23], we have shown that Web services can step in to fill this gap. Web services lend themselves to the extension of knowledge bases, because they deliver structured data. This eliminates the need for noisy information extraction techniques. Furthermore, there are Web services that offer a wide repertoire of data of good quality, well maintained and up to date. This makes Web services an interesting device for complementing knowledge bases. The ANGIE system incorporates Web services as follows: When a user asks a query, ANGIE tries to find the answer in the local knowledge base and resorts to Web services whenever the local knowledge base is not sufficient. ANGIE composes Web services and data from the local knowledge base on the fly, so that the user does not notice that some of the data was not present in the knowledge base before. For example, assume that the user asks for all songs by Canadian singers:

$$\mathsf{nationality}(?x,\mathsf{Canadian}),\ \mathsf{sings}(?x,?y)$$

Then ANGIE could find, e.g., the bindings for $?x$ and some bindings for $?y$ in the local knowledge base, and more bindings for $?y$ by calling an external Web service. One possible answer is ?x=LeonardCohen, ?y=Halleluja.

This approach, however, is bound by the limitations of the APIs of the Web services. For example, there may be a Web service W that returns all songs for a given singer, but no Web service that returns all singers of a

given song. Then, the system cannot directly use W to answer a query for all singers who covered the song Halleluja. W does contain all the necessary information, but its API just does not allow us to query it in the right way. We call this the problem of *Web service asymmetry*.

Formally, a relation R is asymmetric with respect to a Web service API if it allows querying for one argument of R but not for the other one. Even among the most prominent data providers, many relations are asymmetric. We have examined the services isbndb.org, librarything.com, and abebooks.com for books, internetvideoarchive.com for movies, musicbrainz.org, last.fm, discogs.com, and lyricWiki.org for music. Figure 1.1 lists relations in these services that can be queried for the second argument, but not for the first.

citizenOf($pers$,$country$)	rating($movie$,x)
bornIn($pers$,$year$)	graduatedFrom($pers$,$univ$)
livesIn($pers$,$place$)	published($book$,$year$)
hasWon($pers$,$award$)	publishedBy($book$,$editor$)

Figure 1.1: Asymmetric relations in popular Web services.

One naive way of dealing with asymmetric Web services is to try out all possible input values until the Web service delivers the desired output value. For example, to find all singers of Halleluja, we can call the Web service W with all singers we know of, and remember those for which the Web service returns Halleluja. Obviously, this approach quickly becomes infeasible. The first limitation is runtime, with Web service calls taking up to 1 second to complete. Trying out the thousands of singers that a knowledge base such as YAGO [28] contains could easily take hours. The second limitation is the data provider itself, which most likely restricts aggressive querying from the same IP address.

The problem is even more challenging, because it is not trivial to determine whether Web service asymmetry is going to be a problem for a given query or not. Assume, e.g., a Web service sings, which, given the id of a singer, returns his name and his songs. This Web service is asymmetric, because it does not allow using the name of the singer as an input. That means that if the user asks for all the songs by Leonard Cohen, sings cannot be called directly. However, this asymmetry does not pose a problem, if there is another service getId, which, given a singer name, returns his id. We can first call getId(Leonard Cohen) and then feed the result into sings. Thus, we can always try *rewrite* the query so that it avoids an asymmetric call. In general, however, there can be infinitely many rewritings for a given query under a given set of services [20]. This means that the system might spend an infinite amount of time searching for an alternative rewriting without asymmetry.

This leaves us with Web service asymmetry as a problem that does not only drastically restrict the queries we can answer, but that is also not obvious to detect. This limitation applies not just to ANGIE, but to any system that wishes to answer arbitrary queries by combining Web services. Query rewriting approaches that are designed for views with limited binding patterns do not take into account such asymmetry. This is because their goal is to compute the maximal number of answers with no bound on the number of service calls. In order not to miss answers they produce *all* possible rewritings. When dealing with Web services, in contrast, one usually has a limited number of calls and cannot afford trying out all rewritings.

Contribution. In this paper, we propose a solution to the problem of Web service asymmetry. We propose to use information extraction on the fly to "guess" the right input values for the asymmetric Web services. For example, to find all singers of Halleluja, we issue a keyword query "singers Halleluja" to the search engine. We extract promising candidates from the result pages (say, Bon Jovi, Espen Lind, and Elvis Presley). Next, we use the existing Web service to validate these candidates. In the example, we would ask W whether Bon Jovi, Espen Lind, or Elvis Presley sang Halleluja. This confirms the first two singers and discards the last one. This way, we can use an asymmetric Web service as if it allowed querying for an argument that its API does not support.

More precisely, our scenario is as follows: We are given an RDF knowledge base and a set of pre-specified Web services and a limited number of calls. As in [23], our goal is to answer a SPARQL query on the knowledge base by combining local knowledge and results from Web service calls. Our paper makes the following contributions:

- A precise characterization of a class of queries which cannot be processed or cannot be processed in practice by existing mediator-based solutions for Web services. This includes an algorithm to detect such queries.
- A novel approach to answer such queries, where input values for the Web services are extracted on-the-fly from Web pages found by keyword queries. This includes information extraction algorithms that are tailored to our task, considering both full text and HTML tables.
- An experimental evaluation of our approach, for a representative set of queries, using real settings. We integrate the APIs of a variety of high-quality Web services, and compare our approach to existing query answering algorithms.

Our methods are fully implemented in the SUSIE[1] system, which is integrated

[1]Search Using Services and Information Extraction

with our previous work ANGIE [23]. The rest of this paper is structured as follows: Section 2 discusses related work. Section 3 presents our computational model. Section 4 explains how asymmetric relations can be detected and Section 5 explains how they can be treated using information extraction. Section 6 explains the architecture of our system, before Section 7 presents experiments and Section 8 concludes.

2 Related Work

Query Answering. We deal with knowledge bases that are extended on the fly during query evaluation by information from Web services – a scenario that was introduced in ANGIE [23]. We consider a subset of the SPARQL query language where the queries can be written as conjunctive queries and where the Web services can be seen as parametrized queries expressed using the same language.

As values for the inputs are required in order to execute a Web call, a Web service can be seen as a view with limited access patterns [25]. The problem of answering queries using views with limited access patterns was first studied by Rajamaran, Sagiv, and Ullman in [25]. The approach consists in rewriting the initial query into a set of queries where some are executed at the remote data sources publishing the Web services. In [25], the authors show that for a conjunctive query over a global schema and a set of views over the same schema, determining whether there exists a conjunctive query plan over the views that is equivalent to the original query is NP-complete in the size of the query. The result is based on the observation that one may just keep the subgoals that either are mapped to one of the subgoal of the query, or provide an initial binding for one of the variables of the queries. Hence, the size of a query plan can be bounded by the size of the query. However, although the query plan consists of a bounded number of views (functions), there may be *no* bound on the length of the recursive chains of function calls [20]. This has drastical impact on the response time. The same problem has been studied in an extended setting where constraints are added to the set of views [8].

There are two main differences between the problem we address in this work and the problem of answering queries using views [25]. Unlike query answering approaches that aim at computing the set of all logical expressions that might return answers (maximal contained rewritings), we consider a limited number of calls since the Web service providers limit the number of calls that a client is allowed to issue. This changes the approach to solve the

problem. In order to insure the completeness, solutions based on Datalog-bottom up [10, 11, 17] are preferred over top-down Datalog. However, in our setting it is simply impossible to enumerate all Web service results, because most Web services restrict the number of calls coming from an IP address.

Secondly, unlike query rewritings approaches and unlike our work [23], the goal of this work is to detect the class of impractical query evaluations and to propose an alternative to solve them. This alternative consists in suggesting the introduction of new functions. Furthermore, by detecting the class of impractical evaluations, we can prioritize them. This contrasts our work with query rewriting solutions that treat all possible rewritings with answers as equally important in the evaluation. In order to prioritize the calls, we introduce a condition and show that the input values satisfying this condition are better candidates than arbitrary values.

Information Extraction. Quite a number of approaches [3, 28, 34] focus on extracting knowledge from Wikipedia. Others extract knowledge from Wikipedia for query answering on the fly (e.g. [31]). The goal of the present work, however, is to go beyond Wikipedia and tap Web services. Therefore, we regard IE as complementary to our work. Our work uses IE only as a vehicle to find candidate entities that can be submitted to Web services. We briefly review some prominent IE approaches and show where our scenario differs, referring the reader to [26] for a more comprehensive overview of the field.

Named Entity Recognition (NER) approaches such as [35, 21, 7] aim to detect interesting entities in text documents. They can be used to generate candidates for SUSIE. The first approach implemented in this work is a particular simple instantiation of NER, detecting only entities that are known to the knowledge base – thus circumventing the classical challenges in NER.

A second approach we consider is extracting structured information from Web tables. Unlike [5, 12], our algorithm is not limited to HTML lists and tables, but rather detects arbitrary repetitive structures that could contain tabular information. [15] present a slightly different approach based on the visual features of a page.

An array of solutions such as Wrapper Induction [19], fact extraction [1, 4, 29] or entity extraction [7, 36] could be also considered, but they are less practical in our scenario since they require training data.

Complementary Work. In the present work, we assume that the mapping between the schema of the Web Services and the schema of the knowledge base is given. The (semi) automatic creation of schema mappings has been addressed in a large corpus of works, as detailed in the survey [6]. Furthermore, we are not concerned with entity disambiguation and data fusion in this paper. Data fusion is an important component of our system, and it has

been vividly addressed in previous work (e.g., [2]).

Other Web service related problems. A number of works address the problem of automatic composition (or orchestration) of the Web services carrying out complex interactions between Web applications [9]. Other work concerns the composition of Web services that can answer a parameterized user query [30], or return objects of a given type [24]. These approaches do not address the problem of Web service asymmetry.

In contrast to the mash-up approaches [27, 16], our system acts like a mediator system, where the query dynamically combines data from local and external sources, on demand.

3 Computational Model

This section will introduce the problem of query answering on knowledge bases with Web services [23].

Semantic Graph. In tune with recent work [28, 3, 18], we represent our knowledge base in the RDF standard [32] as a *semantic graph*. A semantic graph over a set Ent of entities and a set $Rel \subset Ent$ of relation names is a graph $G \subset Ent \times Rel \times Ent$. Thus, a semantic graph is a set of triples, where each triple expresses a fact. We denote a triple by $R(x, y)$, where R is the relation and x, y are the participating entities. Figure 3.1 shows an excerpt of a semantic graph. In RDFS, there is a distinction between individual entities (such as Leonard Cohen) and class entities (such as the class Singer). Individuals are linked by the type relationship to their class. For example, Leonard Cohen is linked to the class Singer by an edge type(Leonard Cohen, Singer). The classes themselves form a hierarchy. More general classes (such as Person) include more specific classes (such as Singer). This hierarchy is expressed in the semantic graph by edges with the subclassOf relationship, e.g. subclassOf(Singer, Person).

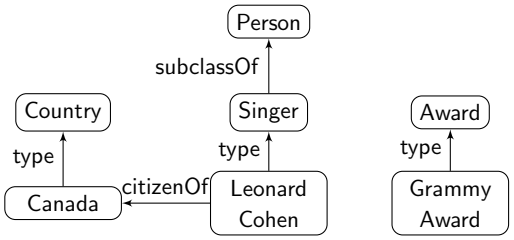

Figure 3.1: A semantic graph fragment.

Query Language. As query language, we consider a subset of the standard RDFS query language SPARQL [33]. Technically, a query over a set of variables Var for a semantic graph $G \subset Ent \times Rel \times Ent$ is a connected semantic graph $Q \subset (Ent \cup Var) \times (Rel \cup Var) \times (Ent \cup Var)$.

Figure 3.2 (left) shows two sample queries: The query Q_1 asks for citizens of Canada. Q_2 asks for citizens of Canada who won the Grammy Award.

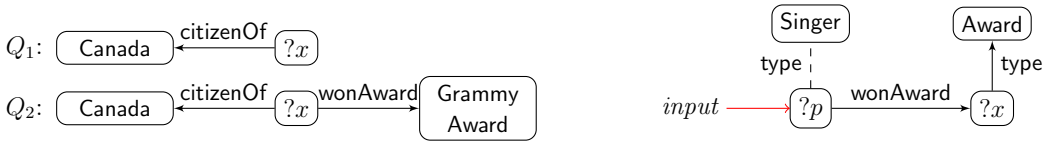

Figure 3.2: Two sample queries (left) and the function $\mathsf{getAwards}^{bf}(?p, ?x)$ (right).

The *answer to a query* Q is a graph homomorphism $\sigma : Q \to \sigma_Q \subseteq G$ that preserves the entity and relation names, and substitutes the variables in Q with entities and relationships names from G. In the example, an answer to Q_1 on the sample semantic graph is the substitution $\sigma = \{?x \to \mathsf{Leonard\ Cohen}\}$, because $\sigma(Q_1)$=$\{\mathsf{citizenOf(Leonard\ Cohen,\ Canada)}\}$ is a sub-graph of the semantic graph. Query Q_2 does not have an answer in the sample graph, because the fact that Leonard Cohen won the Grammy Award is not known to the knowledge base.

Functions. A Web service is an API that can be called over the Internet. Given some input values, it returns as output a semi-structured document, usually XML. Using existing tools [13], mappings can be predefined in the system so that the XML fragments in the results can be translated to RDF-style graphs in the schema of the knowledge base. This allows us to see a Web service as a function on RDF graphs, as follows.

A *function* (of a Web service) is a query, where the set of variables is partitioned into input variables. The input variables have to be bound before the function can be called.

Example. The function getAwards in Figure 3.2 (right) takes as input a value for $?p$ and returns the corresponding awards (binding values for $?x$).

Views with binding patterns The functions can be seen as views with binding patterns [25]. To compare to the existing techniques we revert, when necessary, to the Datalog notation and terminology where a conjunctive query has the form:

$$q(\bar{X}) \leftarrow r_1(\bar{X_1}), r_2(\bar{X_2}), \ldots r_n(\bar{X_n})$$

where q and $r_1, r_2, \ldots r_n$ are predicate names. The predicate names refer to database relations. The atom $q(\bar{X})$ is called the head of the query, and refers to the answer relation. The tuples $\bar{X_1}, \bar{X_2}, \ldots \bar{X_n}$ contain either variables or constants. The query must be *safe*, i.e., $\bar{X} \subseteq \bar{X_1}, \bar{X_2}, \ldots \bar{X_n}$ (every variable in the head must also appear in the body).

In order to model the input and the output parameters of the views, adornments attached to queries have been introduced in [25]. If the head of the query has n attributes, then an adornment consists of a string of length n composed of the letters b and f. The meaning of b is that a binding value *must* be provided for the variable in that position. For example, the function in Figure 3.2 (right), can be written as follows:

$$\mathsf{getAwards}(?x,?y)^{bf} \leftarrow \mathsf{wonAward}(?x,?y),\ \mathsf{Singer}(?x),\ \mathsf{Award}(?y)$$

where the adornment bf says that $?x$ must be bound (input variable) and $?y$ is free (output variable).

4 Answering queries

For a given SPARQL query formulated in terms of the YAGO schema, the goal is to output a query expression formulated in terms of data sources. The resulting expression is called a *rewriting*. Since, in SUSIE, data sources are accessible by function calls, the rewriting is a sequence of function calls. As we assume limited access to data sources, the focus is on efficiently ordering the list of calls. For the beginning, we consider the problem of computing answers for a single query edge, where one node value is known. Even this simple case may require call compositions. In [20], it is shown that there may be *no* bound on the size of the rewriting. As an example, consider the following functions (adapted from [10]):

$$\mathsf{getAwards}(?s,?a)^{bf} \leftarrow \mathsf{wonAward}(?s,?a),\mathsf{Singer}(?s),\mathsf{Award}(?a)$$
$$\mathsf{getSimTo}(?s_1,?s_2)^{bf} \leftarrow \mathsf{similarTo}(?s_1,?s_2),\mathsf{Singer}(?s_1),\mathsf{Singer}(?s_2)$$

and let Q be the following query:

$$\mathsf{Q}^{bf}(?a) \leftarrow \mathsf{wonAward}(\mathsf{Leonard\_Cohen},?a)$$

For this query, the call getAwards(Leonard_Cohen) will return the bindings for $?a$. Now, consider the following query:

$$\mathsf{Q}^{fb}(?p) \leftarrow \mathsf{wonAward}(?p,\ \mathsf{Grammy\_Award})$$

Since the function $\mathsf{getAwards}(?s,?a)^{bf}$ requires its input to be bound it cannot be called directly. One way to get solutions is to issue a call to the function getAwards for all the singers in the local base. By calling the function getSimTo, more singer names can be found. Even more singer names can be obtained by recursively calling getSimTo. Thus, the role of getSimTo is simply to generate all elements in the class Singer (that can be obtained by the means of function calls). Indeed, for such cases, in [10], the authors propose to construct a new intermediate relation Singers whose extension is the set of all singers that are reachable by computing chains of getSimTo calls.

In this scenario, the chance that one service call returns a desired entity is proportional to the ratio of elements of the domain that have the desired property. In our example, the chance that a single service call returns a singer that won the Grammy Award is the number of singers that won the Grammy Award divided by the total number of singers. This is usually an insignificant number (we show in the experiments that this ratio is around 1%). This means that most service calls will happen in vain (on average, 99% of the service calls are in vain). This query evaluation strategy is complete, but it is not feasible in reality, because the Web service provider restricts the number of permitted calls. In the example, if we limit the calls to 15, we will most likely not get any answer.

If appropriate functions are available, there can be a rewriting that does not enumerate the domain. For example, if we had the function getPersonForAward, then there would exist a rewriting of the above query that does not enumerate the domain. If all possible rewritings of a query edge under a given set of functions require enumerating a domain, we call the edge an *impractical* query edge. Since the number of rewritings can be infinite, it is not trivial to determine whether a query edge is impractical. The goal of the present section is to develop an algorithm that can detect the impractical edges of a query. Section 5 will then discuss how to handle impractical query edges.

The remaining of this section we introduce the concept of recursive rewritings and we show that only a finite number of rewritings need to be considered in order to determine whether a query edge is impractical. We also consider rewritings for entire conjunctive queries.

4.1 Recursive Rewritings

We consider a single query edge q. Without loss of generality, we assume that the first argument of q is bound, i.e. $q = R^{bf}(c, ?y)$. Our goal is to determine whether q is impractical, i.e., whether all possible rewritings of q will require the enumeration of a domain. For simplicity, let us consider first the case where functions have only one input. We distinguish 4 cases:

(A) Simple case: There is a function f of the form

$$f^{bf\cdots}(?x, ?y, ...) \leftarrow R(?x, ?y), ...$$

where f may contain more than one variable and its body may contain more than one relation. In this case, q can be answered by calling f. Hence, q is not impractical.

(B) Recursive case: Case (A) does not apply, and there is a function f that has a path from the input value $?x_1$ to the query edge that does not contain the edge matching the query $R(c, ?y)$:

$$f^{bf\cdots}(?x_1, ...) \leftarrow R_1(?x_1, ?x_2), ..., R_{n-1}(?x_{n-1}, ?x_n), R_n(?x_n, ?x), R(?x, ?y)...$$

In this case, we can answer q by calling f, if we find a value for $?x_1$ that binds $?x$ to c. Finding this value means first finding an $?x_n$, such that $x = c$ and thus $R_n(?x_n, c)$. Then one has to find an appropriate $?x_{n-1}$ and so forth. Hence, the problem is recursively reduced to determining whether these query edges are impractical. Note that there can be multiple cases (B) for the same function f, because there can be multiple paths from $?x_1$ to the query edge.

(C) Impractical case: Cases (A) and (B) do not apply, but there exist functions that can return relationships R. Such functions must have one of the following forms:

(1) $f^{fb\cdots}(?x, ?y, ...) \leftarrow R(?x, ?y), ...$
(2) $f^{bf\cdots}(?x_1, ...) \leftarrow R_1(?x_1, ?x_2), ..., R_{n-1}(?x_{n-1}, ?x_n)\ R_n(?x_n, y), R(?x, ?y)...$

The first form is the impractical case that requires enumerating the entire domain for the inputs $?x$. The second form induces a vicious circle: In order to compute the good inputs $?x_1$ that generate solutions for $R(?x, ?y)$, we would have to find $?y$ first so that $R(c, ?y)$. Thus, the recursion does not reduce the problem.

(D) Impossible case: None of the functions returns relationships R. In this case, the query edge cannot be answered at all.

Thus, a naive algorithm for determining whether a query edge is impractical consists of recursively checking all 4 cases. This will build up a tree structure of rewritings. If all rewritings involve case (C), then the edge is impractical. Unfortunately, however, there may be an infinite number of rewritings. We will therefore try to cut some of the branches of the rewriting tree. For this purpose, let us look at the case (B). We define the *query precondition*:

Definition 1 (Query Precondition) *A precondition for a query $q = R^{bf}(c, ?y)$, and a function f returning R relationships is a query $Prec_{q,f}$ that consists of a path in f from the input value of f to the node that matches c.*

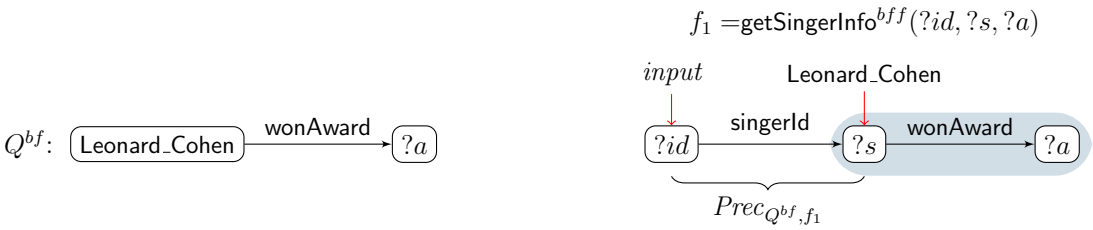

Figure 4.1: Query precondition example

For instance, consider the query Q^{bf} and the function f_1 =getSingerInfo in Figure 4.1. The precondition for Q^{bf} and getSingerInfo is the query:

$$Prec_{Q^{bf},f_1}(\text{Leonard\_Cohen},?id)^{bf} \leftarrow \text{singerId}^{fb}(?id,\ \text{Leonard\_Cohen})$$

Every possible query precondition constitutes one way of answering the query edge with the function. That is, every possible query precondition will give rise to one instance of case (B) above. If a function has multiple input values, then the notion of the query precondition needs to be modified to include *all* input nodes. All other considerations still apply also in the general case.

4.2 Avoiding Infinite Rewritings

Let us consider computing rewritings for a given single query edge $q = R^{bf}(c, ?y)$. q will fall into one of the above cases, (A), (B), (C), or (D). In the cases (A), (C), and (D), no further rewritings are necessary. Let us therefore consider case (B). A rewriting corresponding to the case (B) will, in the next recursive step, give rise to finding the rewritings of $e = R^1_n(x^1_n, c)^{fb}$. Figure 4.2 illustrates this scenario. If this edge e has ever appeared in the rewriting before, with the same relation name and with the first argument free and the second one bound, but with any constant in place of c, then e is impractical iff the other edge is impractical. Hence we do not have to deal recursively with the same problem again. The same applies if an edge appears with the inverse relation and the first argument bound and the second one free. Therefore, we do not need to rewrite e at all. If there is no such similar edge, we have to rewrite e. For this new sub-goal, we can have again one of the cases (A), (B), (C) or (D). In case (A), (C), or (D), we are done and no more rewritings are required. Let us therefore consider the case (B) for $R^1_n(x^1_n, c)^{fb}$. This will give rise to a new sub-goal $R^2_n(x^2_n, c)^{fb}$. Again, this edge does not require any rewriting if a similar edge exists already somewhere. Otherwise, the edge will again fall into one of the cases (A), (B), (C)

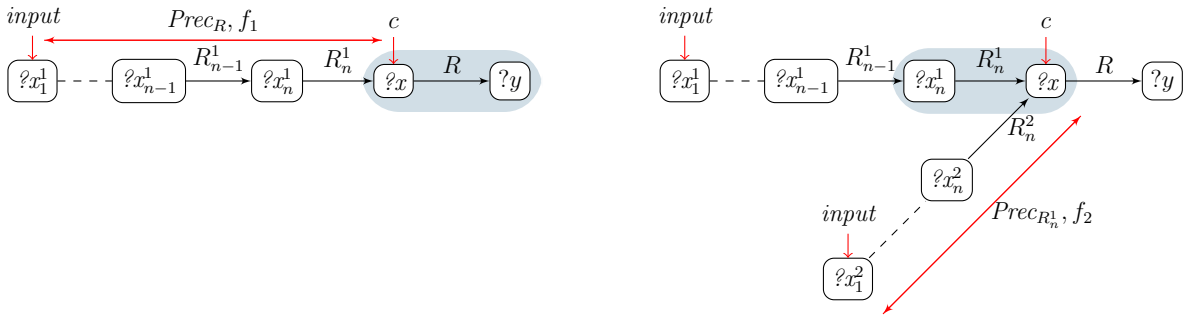

Figure 4.2: Recursive steps. Step 1 (left): f_1 might return $R(c, ?y)$ if answers for $Prec_{R,f_1}$ can be computed. Step 2 (right): f_2 might return $R_1^1(x^1, c)$ if answers for $Prec_{R_n^1,f_2}$ can be computed.

or (D). This procedure continues until case (B) has been encountered for all relations with both possible binding distributions, bf and fb.

This means that case (B) can only appear $2n$ times, where n is the number of relation names. Since the number of relation names is finite, the number of functions is finite and the number of call preconditions is finite, this means that the total number of required rewritings is finite, too. This means that it can be determined in finite time whether a query edge is impractical.

4.3 Query evaluation

Consider the evaluation of a query Q that contains at least one constant c. Since Q is a connected graph, there is at least one left-to-right evaluation

$$R_1^{bf}(c, ?x_1) \wedge R_2^{bf}(?x_1, ?x_2) \ldots \wedge R_n^{bf}(?x_{n-1}, ?x_n)$$

where $R_1, \ldots, R_n$ represent the query edges in Q and c is a constant. We consider the computation R^{bb} as a particular case of the computation of R^{bf}. At step i, answers for $R_1^{bf}(c, ?x_1) \wedge R_2^{bf}(?x_1, ?x_2) \ldots R_i^{bf}(?x_{i-1}, ?x_i)$ are produced. Hence, at step i the variables $?x_1, ?x_2 \ldots ?x_i$ are bound to values. These values could be further used if necessary as inputs to Web calls in order to evaluate $R_{i+1}^{bf}(?x_i, ?x_{i+1})$.

Thus, answering the query is possible if there exists a left-to-right evaluation that (i) respects the input conditions and that (ii) does not contain an impractical edge. The first requirement can be checked by the known algorithm [23]. The second requirement can be checked by the recursive evaluation given in Section 4.2. This algorithm is complete (i.e., it will deliver all possible answers to the query) iff the functions allow computing the

entire set of query preconditions. This is unlikely in practice, as it is with all data integration approaches.

Now let us consider the case where the query contains an impractical query edge. This is the case where all query rewritings contain (C), i.e., where we need the inverse of an existing function f. In this case, we propose to introduce the inverse function f^{-1}. Thereby, the query edge is no longer impractical. We will discuss in the next Section how we can introduce f^{-1}.

5 Bindings from the Web

We will now discuss how we can construct the inverse function of an asymmetric Web service, in order to eliminate an impractical edge in a query rewriting. As an example, assume that the impractical query edge is wonAward($?x$, Grammy Award). Assume that we only have the inverse function wonAward^{bf}($?x$, y), which requires a person as input and returns the awards of that person as output.

SUSIE will construct a virtual function wonAward^{fb}($?x$, y). Whenever this function is called, SUSIE will do the following: It will issue calls of the form "List of person won y" to a Web search engine such as Google. Then, SUSIE will extract possible candidates for x from the Web page. Last, SUSIE will call the real Web service wonAward^{bf}($?x$, y) to check which candidates return the desired y. These candidates are returned as output to the call to the virtual function. Thereby, the virtual function acts just like a real Web service function.

We will now discuss two subtasks of this endeavor: (1) the task of finding suitable Web pages and (2) the task of extracting candidates from the Web pages. Even though a high precision and a high recall are desirable for the two sub-tasks, they are not strictly necessary. If the precision is low, this will result in more Web service calls, but not in diminished precision of the final query answers because all answers are checked by the Web service. If recall this low, this will result in fewer answers. Fewer answers, however, are better than trying out all possible input values for the function, which may result in no answer at all due to the limited call budget.

Web Search. We first note that a relation in RDF typically comes with a domain and a range (this is the case in YAGO [28]). Thereby, we already know the type of the candidate values (the *target type*). In the example of wonAward, we know that the domain is Person and hence we know that we are looking for instances of the class Person. Since the names of the relationships, as defined in the database schema, are usually less suitable to be used in keyword search queries, we manually map each relation name to

a *search string*. For example, the relation wonAwards with a second bound argument y is mapped to the string *"List of persons won award y"*. If the first argument x is bound, we map it to *"x list of awards"*.

We submit the query to a common Internet search engine and collect the top ten result Web pages. It is close to impossible to evaluate systematically with the dozens of relationships and millions of constants that YAGO contains that the Web pages obtained in this manner contain the good candidate entities. Still, experience from our experiments indicates that information of common interest is often publicly available on the Web. Furthermore the same information appears in several of the returned pages. This increases the chances of an IE algorithm to extract the required entities.

Information Extraction. Once the Web pages have been retrieved, it remains to extract the candidate entities. Information extraction is a challenging endeavor, because it often requires near-human understanding of the input documents. Our scenario is somewhat simpler, because we are only interested in extracting the entities of a certain type from a set of Web pages. We have implemented two simple yet effective information extraction algorithms as a proof of concept.

String Matching Algorithm. For this algorithm, we are only interested in entities that are already known to the knowledge base. This is a reasonable focus in our case, because our knowledge base YAGO feeds from Wikipedia and thus already contains a large number of entities of common interest. Using a trie [14] that contains all entities of the target type, we extract all entities of the target type from the Web pages. This processing can be done in time $O(n)$ in the best case and in time $O(m \cdot n)$ in the worst case, where n is the total number of characters in the Web pages and m is the number of characters in the longest entity name. These entities are returned as output of the virtual function.

Quasi-Table Extraction. The String Matching Algorithm has the disadvantage that it can only find entities that appear in YAGO. If we wish to venture beyond this limitation and find new entities, we may exploit that many result Web pages will have a structured form. Typically, tables represent a natural way to organize sets of relationships in Web pages. However, as shown in [15], only a small fraction of the Web tables are encoded using the `<table>` markup in HTML. In many cases, they are encoded using lists or loosely repetitive structures. We call such structures *quasi-tables*.

One way of detecting many quasi-tables is identifying structures of repetitive rows, where each row contains items that are separated by special strings or tags that re-appear in each row. Furthermore, the items in one column are of the same syntactic type (numbers, strings or dates). This definition subsumes standard tables and standard lists. We have developed an algorithm

that can detect such structures. By comparing the elements of each column with the instances of the target type in YAGO, our algorithm finds the column that constitutes most likely the answers to the query. The elements of this column are returned as outputs of the virtual function.

We note that these are just two straight-forward implementations. They can be replaced by more sophisticated implementations [12, 5].

6 System architecture

The overall architecture of our system is illustrated in Figure 6.1. The system uses the existing YAGO knowledge base [28], which consists of 2 million entities and 20 million facts extracted from encyclopedic Web sources. In addition, we extended the knowledge with a built-in collection of function definitions for the following Web services: `MusicBrainz`, `LastFM`, `LibraryThing`, `ISBNdb`, `AbeBooks`, and `IVA` (Internet Video Archive). In our envisioned long-term usage, the function definitions would either be automatically acquired from a Web-service broker/repository or they could be semi-automatically generated by a tool, e.g., [13].

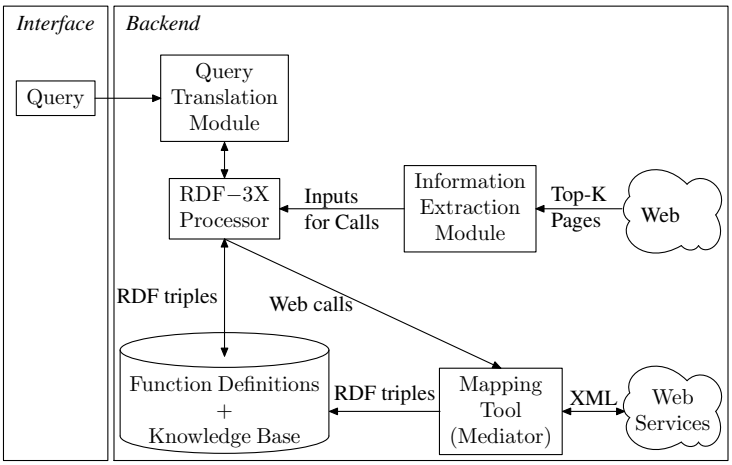

Figure 6.1: System architecture.

The core of the present work is implemented in the Query Translation Module. This module takes as input a user query, and translates it into a sequence of function compositions. This module also detects impractical query edges. For each such edge, it creates a virtual Web service.

The translation module continuously sends SPARQL queries with embedded Web service calls to the RDF-3X processor [22]. The RDF-3X processor has been modified to accommodate Web service calls. It is responsible for scheduling the execution of the function calls, and integrating the results in the processing of the input query. The calls are executed via the *Mapping Tool*, which is in charge of remote invocation. The Mapping Tool also translates the answer of the Web service call back into RDF, according to a predefined mapping. It responds to the processor with the list of RDF triples representing the answers of the calls. The RDF-3X processor combines the triples from the local knowledge base and the triples received from the mapping tool to produce a uniform output. The query translation and the query execution are interleaved.

7 Performance evaluation

We conducted 2 types of experiments. We first evaluate exclusively the performance of the information extraction algorithms. Then, we evaluate the actual performance of SUSIE on real-world queries.

7.1 Information Extraction

To evaluate our IE algorithms independently of the queries and the function compositions, we constructed test sets of Web pages and extracted the target entities manually from these pages. We then ran the extraction algorithms and measured their performance with respect to the manually extracted gold standard. We targeted three query types: Queries that ask for actors with a certain birth year, for actors with a certain nationality and for authors who received a certain prize. For each query type, we choose 10 arbitrary property values (10 birth years, 10 nationalities and 10 literature prizes). For each property value, we generated the keyword query that SUSIE would generate for the query, sent it to Google and retrieved the top 10 pages. For example, for the query "actors born in 1970", we issued the query "List of actors born in 1970". This gave us 100 pages for each test set. The pages are quite heterogeneous, containing lists, tables, quasi-tables and full-text listings of entities. We then ran the extraction algorithms. Figures 7.1 and 7.2 show our results. We compare the results to the results that we would achieve without information extraction. For this purpose, we report the number of entities of the target type in the YAGO database that have the desired property. For example, for the query "actors born in 1970", we report the proportion of actors in YAGO that are born in 1970 (among those actors that have a birth date).

The precision and recall are nearly always in a healthy range between 30% and 75%. A precision of 30% means that, for every 3 queries that are sent to the Web service, 2 are sent in vain. We find this acceptable. A recall of 30%

Country	#E	SMA		QTE		YAGO
		Prec	Rec	Prec	Rec	Prec
Australia	15	37 %	82 %	51 %	66 %	11%
Canada	5	28 %	92 %	40 %	50 %	20%
England	46	46 %	85 %	71 %	74 %	0 %
France	153	48 %	42 %	50 %	64 %	2 %
Germany	45	50 %	57 %	51 %	99 %	2 %
Greece	26	38 %	58 %	2 %	14 %	0 %
Italy	138	29 %	54 %	42 %	59 %	0 %
Mexico	25	44 %	52 %	51 %	78 %	0 %
South Africa	12	29 %	76 %	29 %	63 %	0%
Spain	24	54 %	63 %	67 %	94 %	0 %
	47	38 %	65 %	46 %	63 %	3.5 %

All numbers averaged over 10 pages per line #E = Average number of entities per page

Figure 7.1: Information Extraction Results for "Actors of nationality X"

Award	#E	SMA		QTE		YAGO
		Prec	Rec	Prec	Rec	Prec
Franz Kafka	2	25 %	73 %	13 %	34 %	N/A
Golden Pen	9	36 %	33 %	29 %	56 %	N/A
Jerusalem	6	23 %	52 %	69 %	24 %	N/A
National Book	69	38 %	59 %	45 %	76 %	0.9 %
Nobel Prize	44	41 %	29 %	46 %	40 %	2.9 %
Phoenix	4	47 %	71 %	18 %	76 %	N/A
Prix Decembre	4	29 %	6 %	18 %	25 %	N/A
Prix Femina	21	31 %	13 %	32 %	32 %	0.6 %
Prix Goncourt	73	63 %	46 %	7 %	1 %	1.12%
Pulitzer	42	78 %	79 %	60 %	46 %	2.0 %
	27	43 %	44 %	34 %	35 %	1.5%

Figure 7.2: IE Results for "Authors who won prize X"

Year	#E	SMA		QTE		YAGO
		Prec	Rec	Prec	Rec	Prec
1940	2	2 %	73 %	1 %	80 %	0.8 %
1945	1	2 %	96 %	1 %	100 %	1.0 %
1950	2	2 %	81 %	1 %	83 %	1.2 %
1955	2	6 %	39 %	3 %	56 %	1.2 %
1960	18	12 %	60 %	6 %	72 %	1.3 %
1965	8	17 %	72 %	14 %	71 %	1.5 %
1970	4	20 %	96 %	1 %	66 %	1.7 %
1975	2	8 %	91 %	1 %	67 %	1.6 %
1980	6	8 %	52 %	4 %	90 %	1.6 %
1985	2	8 %	56 %	0 %	43 %	1 %
	5	9 %	71 %	3 %	74 %	1.3 %

Figure 7.3: IE Results for "Actors born in year X"

means that we can find one third of the entities that the user is potentially interested in. Only the precision on the birth year queries is disappointing, with values below 10% (Figure 7.1). This is because the Google queries returned lists of all actors, not just of the ones born in a certain year. Thus, the extraction algorithms find far too many irrelevant entities in the pages. The QTE algorithm, with its slightly higher recall, suffers particularly for

No.	Query		Constants
1	type	(?person, Writer)	p
	wonAward	(?person, p)	
2	type	(?person, Writer)	p, c
	wonAward	(?person, p)	
	isCitizenOf	(?person, c)	
3	type	(?person, Writer)	p
	wonAward	(?person, p)	
	wrote	(?person, ?book)	
4	type	(?person, Writer)	p, c
	wonAward	(?person, p)	
	isCitizenOf	(?person, c)	
	wrote	(?person, ?book)	
5	type	(?person, Writer)	p, y
	wonAward	(?person, ?prize)	
	isTitled	(?prize, p)	
	awardedInYear	(?prize, y)	
6	type	(?person, Writer)	p
	wrote	(?person, ?book)	
	wonAward	(?person, ?prize)	
	isTitled	(?prize, p)	
	awardedInYear	(?prize, y)	

No.	Query		Constants
7	type	(?person, Actor)	c
	isFamousActor	(?person, True)	
	isCitizenOf	(?person, c)	
8	type	(?person, Actor)	c
	isFamousActor	(?person, True)	
	isCitizenOf	(?person, c)	
	actedIn	(?person, ?movie)	
9	type	(?person, Actor)	p
	wonAward	(?movie, p)	
	actedIn	(?person, ?movie)	
10	type	(?person, Actor)	p
	wonAward	(?movie, p)	
	producedIn	(?movie, ?country)	
	actedIn	(?person, ?movie)	
11	type	(?person, Singer)	p, y
	sang	(?person, ?song)	
	wonAward	(?person, ?prize)	
	isTitled	(?prize, p)	
	awardedInYear	(?prize, y)	

Figure 7.4: Query templates

the precision. We record this as a case where the information extraction approach is less practical, because the Internet does not provide the lists of entities that the approach needs.

Still, we note that, in all cases, our approach outperforms the naive approach of sending all entities of the target type to the Web service. In general, the proportion of entities that have the desired property is very low. The percentages for writer awards are already an overestimation, because they consider only those writers that did win an award, while many writers do not win any award at all in their life. So let us e.g. assume that 1% of the entities have the desired property. This means that an expected 100 calls would have to be sent to the Web service before finding one of them. This number of calls is already above the budget we are considering, meaning that the user would likely not get any response at all. Thus, even in the cases with lower precision, our approach allows answering queries that would be impossible to answer otherwise.

7.2 Real-world Queries

We compare the SUSIE algorithm with two competitors, the algorithms DF and F-RDF presented in [23]. The DF algorithm implements a Prolog-style backtracking strategy. The F-RDF improves over the DF algorithm since it chooses as bindings for the input parameters values from the local base that satisfy already some of the constrains in the query. Hence, they have

Q	Constants	Empty Database				YAGO Database											
		First		Last		First Result						Last Result					
		SUSIE				SUSIE		F-RDF		DF		SUSIE		F-RDF		DF	
		call	anws	call	anws	call	anws	call	anws	call	anws	call	anws	call	anws	call	anws
1	Nobel Prize in Literature	3	1	55	14	0	103	0	103	0	103	0	103	0	103	0	103
	Golden Pen Award	4	1	16	11	4	1		0		0	16	11		0		0
	Franz Kafka Prize	4	1	8	5	4	1		0		0	8	5		0		0
	American Book Medal	3	1	18	16	3	1		0		0	18	16		0		0
	Jerusalem Prize	3	1	21	11	3	1		0		0	21	11		0		0
2	France, Nobel Prize Literature	2	1	9	5	0	6	0	6	2	6	8	9	90	6	2	6
	UK, Franz Kafka Prize	2	1	2	1	2	1		0		0	2	1		0		0
3	Nobel Prize Literature	43	15	100	198	0	234	0	234	15	234	94	457	120	453	45	235
	Golden Pen Award	18	13	87	228	6	1		0		0	99	226		0		0
	Franz Kafka Prize	19	9	97	132	5	14		0		0	92	181		0		0
	American Book Medal	19	18	97	296	3	3		0		0	111	522		0		0
	Jerusalem Prize	22	6	90	220	4	1		0		0	91	233		0		0
4	France, Nobel Prize Literature	11	12	89	144	0	2	0	2	2	2	107	133	132	74	113	61
	UK, Franz Kafka Prize	3	19	63	79	3	18		0		0	61	70		0		0
5	Nobel Prize Literature 2004	2	1	2	1	2	1		0		0	2	1		0		0
	Golden Pen Award 2006	2	1	2	1	2	1		0		0	2	1		0		0
	Franz Kafka Prize 2006	2	1	2	1	2	1		0		0	2	1		0		0
	American Book Medal	2	1	2	1	2	1		0		0	2	1		0		0
	Jerusalem Prize 1981	2	1	2	1	2	1		0		0	2	1		0		0
6	Nobel Prize in Literature 2004	3	9	51	31	3	9		0		0	51	31		0		0
	Golden Pen Award 2006	3	12	52	57	3	12		0		0	51	64		0		0
	Franz Kafka Prize 2006	3	10	59	61	2	14		0		0	59	89		0		0
	American Book Medal	3	18	75	77	2	139		0		0	85	243		0		0
	Jerusalem Prize 1981	3	16	71	60	2	17		0		0	69	90		0		0
7	United States Of America	5	1	20	7	5	1		0		0	20	7		0		0
	United Kingdom	3	1	26	7	3	1		0		0	26	7		0		0
8	United States Of America	5	46	40	309	5	48		0		0	40	330		0		0
	United Kingdom	3	32	52	213	3	36		0		0	66	234		0		0
9	Academy Award Best Picture	2	4	56	85	2	14		0		0	88	187		0		0
10	Academy Award Best Picture	2	4	16	79	2	5		0		0	94	212		0		0
11	Grammy Awards 2009	69	31	75	110	69	14		0		0	75	377		0		0

Figure 7.5: Results for the SUSIE algorithm

better chances to lead to a solution. The SUSIE algorithm was developed as an extension of the F-RDF algorithm. All experiments use the YAGO [28] knowledge base.

Testbed and Methodology

Evaluated Methods. We have implemented every algorithm as part of the query answering component of our prototype system. The fully functional system is implemented in Java. For all the algorithms, we set the budget to 15 for the number of calls to one service and to 100 for the total number of calls. As performance metrics, we measured the total number of answers output by each algorithm.

Data sources. We ran experiments for two distinct settings. In the first setting, we use an empty knowledge base. We use YAGO only to determine the type of a candidate value. In the second scenario we use the full YAGO knowledge base. We shall see that for a variety of queries the classical an-

swering query approaches produce no answers due to the asymmetry of Web services. We integrated data from different domains via Web services: isbndb.org, librarything.com, and abebooks.com for books, internetvideoarchive.com for movies, musicbrainz.org, last.fm, discogs.com, and lyricWiki.org for music. All these Web sites allow users to query their data through Web services. For each Web service, we manually defined mapping functions from the XML output into the schema of YAGO.

Queries. We selected a variety of query templates, which can be organized in the following classes (Figure 7.4): star queries with constants at the endpoints (Q_1-Q_2, Q_7), star queries with variables and constants at the endpoints (Q_3-Q_4, Q_8-Q_{10}), and chain queries with constants at the endpoints (Q_5-Q_6, Q_{11}).

For every query template, we evaluate a set of similar queries by varying the constants. As we shall see in the next paragraph, such queries are impractical using only Web services and no answers can be obtained if the local base is empty. Most of the queries have different alternative ways of composing function instantiations. Usually, this leads to a high number of Web service calls.

Results. Figure 7.5 shows the results for the queries in Figure 7.4. The table reports the number of answers as well as the number of calls that were necessary in the evaluation for two moments in time: (*i*) when the first results are output (*ii*) when the last result is output. We omit the number of calls for the cases where no answer is ever output. Since all the algorithms use the same number of calls, the total number of answers returned by each serves as comparison metric. For the case where the local base is empty, only the answers for the SUSIE algorithm are reported as all the other algorithms using only the Web service interfaces return no answers. For the second case, the YAGO database offers already some query answers. However, obtaining new answers is in the majority of cases unlikely to happen.

We observe that, for all queries, SUSIE returns more answers than her competitors or an equal number. For instance, for the query Q_4, SUSIE (F-IE) outputs almost twice as many answers than F-RDF outputs with a number of calls that is 30% less than the number of calls used by F-RDF. Figure 7.6 shows the distribution of the results at different moments of the evaluation.

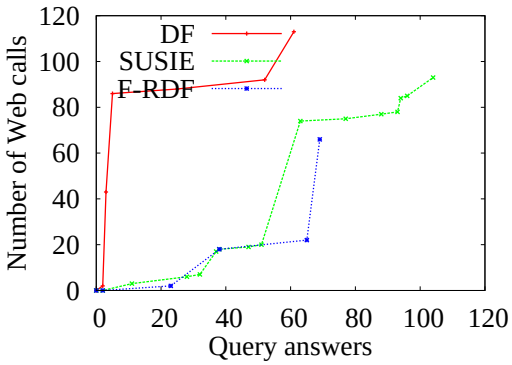

Figure 7.6: Answer distribution for Q_4 with the YAGO knowledge base

8 Conclusion

This paper has introduced the problem of asymmetric Web services. We have shown that a considerable number of real-world Web services allow asking for only one variable of a binary relationship, but not for the other. We have shown how one can determine whether a query will suffer from the problem of Web service asymmetry. We have proposed to use information extraction to guess bindings for the variable and then validate this binding by the Web service. Through this approach, a whole new class of queries has become tractable. We have implemented our system, SUSIE, and showed the validity of our approach on real data sets.

Our current implementation uses naive information extraction algorithms that serve mainly as a proof of concept. Future work will explore new algorithms that could step in. We also aim to automize the discovery of new Web services and their integration into the system. Furthermore, the idea of using information extraction to circumvent API restrictions is not limited to Web services: The exploration of other services with restricted access patterns, such as for example Deep Web forms, could also benefit from guessing input values with information extraction.

Bibliography

[1] E. Agichtein, L. Gravano, J. Pavel, V. Sokolova, and A. Voskoboynik. Snowball: a prototype system for extracting relations from large text collections. *SIGMOD Records*, 30(2), 2001.

[2] A. Arasu and R. Kaushik. A grammar-based entity representation framework for data cleaning. In *SIGMOD*. ACM, 2009.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a Web of Open Data. *The Semantic Web*, 2008.

[4] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *IJCAI*, 2007.

[5] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational web. In *WebDB*, 2008.

[6] N. Choi, I.-Y. Song, and H. Han. A survey on ontology mapping. *SIGMOD Rec.*, 35, September 2006.

[7] H. Cunningham and D. Scott. Software architecture for language engineering. *Nat. Lang. Eng.*, 10(3-4), 2004.

[8] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. *Theor. Comput. Sci.*, 371(3):200–226, 2007.

[9] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web services. In *PODS*, 2004.

[10] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, 1997.

[11] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive query plans for data integration. *J. Log. Program.*, 43(1), 2000.

[12] H. Elmeleegy, J. Madhavan, and A. Y. Halevy. Harvesting relational tables from lists on the web. *PVLDB*, 2(1), 2009.

[13] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications*, 2009.

[14] E. Fredkin. Trie memory. *Commun. ACM*, 3(9), September 1960.

[15] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *WWW*. ACM, 2007.

[16] M. Jarrar and M. D. Dikaiakos. MashQL: a query-by-diagram topping SPARQL. In *ONISW*, 2008.

[17] S. Kambhampati, E. Lambrecht, U. Nambiar, Z. Nie, and G. Senthil. Optimizing recursive information gathering plans in emerac. *J. Intell. Inf. Syst.*, 22(2), 2004.

[18] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and Ranking Knowledge. In *ICDE*, 2008.

[19] N. Kushmerick. *Wrapper induction for information extraction.* PhD thesis, University of Washington, 1997.

[20] C. T. Kwok and D. S. Weld. Planning to gather information. In *AAAI/IAAI, Vol. 1*, 1996.

[21] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*. Morgan Kaufmann Publishers Inc., 2001.

[22] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *PVLDB*, 1(1), 2008.

[23] N. Preda, G. Kasneci, F. M. Suchanek, T. Neumann, W. Yuan, and G. Weikum. Active Knowledge : Dynamically Enriching RDF Knowledge Bases by Web Services. (ANGIE). In *SIGMOD*, 2010.

[24] K. Q. Pu, V. Hristidis, and N. Koudas. Syntactic rule based approach to Web service composition. In *ICDE*, 2006.

[25] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *PODS*, 1995.

[26] S. Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 2(1), 2008.

[27] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh. Damia: data mashups for intranet applications. In *SIGMOD*, 2008.

[28] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A Core of Semantic Knowledge. In *WWW*. ACM Presss, 2007.

[29] F. M. Suchanek, M. Sozio, and G. Weikum. SOFIE: A Self-Organizing Framework for Information Extraction. In *WWW*. ACM Press, 2009.

[30] S. Thakkar, J. L. Ambite, and C. A. Knoblock. Composing, optimizing, and executing plans for bioinformatics web services. *VLDB J.*, 14(3), 2005.

[31] H. L. Wang, S. H. Wu, I. C. Wang, C. L. Sung, W. L. Hsu, and W. K. Shih. Semantic search on internet tabular information extraction for answering queries. In *CIKM*. ACM, 2000.

[32] Word Wide Web Consortium. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 2004-02-10.

[33] World Wide Web Consortium. SPARQL Query Language for RDF (W3C Recommendation 2008-01-15), 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

[34] F. Wu and D. S. Weld. Automatically refining the Wikipedia infobox ontology. In *Proc. of the Int. WWW Conf.*, 2008.

[35] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. 2d conditional random fields for web information extraction. In *ICML*. ACM, 2005.

[36] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. Simultaneous record detection and attribute labeling in web data extraction. In *KDD*, New York, NY, USA, 2006. ACM.