LSVS: Link Specification Verbalization and Summarization

Abdullah Fathi Ahmed[1] Mohamed Ahmed Sherif[1,2] and Axel-Cyrille Ngonga Ngomo[1,2]

[1] Paderborn University, Data Science Group, Pohlweg 51, D-33098 Paderborn, Germany
E-mail: {firstname.lastname}@upb.de
[2] Department of Computer Science, University of Leipzig, 04109 Leipzig, Germany
E-mail: {lastname}@informatik.uni-leipzig.de

Abstract. An increasing number and size of datasets abiding by the Linked Data paradigm are published everyday. Discovering links between these datasets is thus central to achieve the vision behind the Data Web. Declarative Link Discovery (LD) frameworks rely on complex Link Specification (LS) to express the conditions under which two resources should be linked. Understanding such LS is not a trivial task for non-expert users, particularly when such users are interested in generating LS to match their needs. Even if the user applies a machine learning algorithm for the automatic generation of the required LS, the challenge of explaining the resultant LS persists. Hence, providing explainable LS is the key challenge to enable users who are unfamiliar with underlying LS technologies to use them effectively and efficiently. In this paper, we address this problem by proposing a generic approach that allows a LS to be verbalized, i.e., converted into understandable natural language. We propose a summarization approach to the verbalized LS based on the selectivity of the underlying LS. Our adequacy and fluency evaluations show that our approach can generate complete and easily understandable natural language descriptions even by lay users.

Keywords: Open Linked Data, Verbalization, Link Discovery, Link Specification, NLP, Text Summarization

1 Introduction

With the rapid increase in the number and size of RDF datasets comes the need to link such datasets. Declarative Link Discovery frameworks rely on complex Link Specification to express the conditions necessary for linking resources within these datasets. For instance, state-of-the-art LD frameworks such as Limes [13] and Silk [9] adopt a property-based computation of links between entities. For configuring LD frameworks, the user can either (1) manually enter a LS or (2) use machine learning for automatic generation of LS.

There are a number of machine learning algorithms that can find LS automatically, by using either supervised, unsupervised or active learning. For example, the Eagle algorithm [15] is a supervised machine-learning algorithm able to learn LS using genetic programming. In newer work, the WOMBAT algorithm [19] implements a positive-only learning algorithm for automatic LS finding based on generalization via an upward refinement operator. While LD experts can easily understand the generated LS from such algorithms, and even modify if necessary, most lay users lack the expertise to proficiently interpret those LSs. In addition, these algorithms have been so far unable to explain the LS they generate to lay users. Consequently, these users will face difficulty to i) assess the correctness of the generated LS, ii) adapt their LS, or iii) choose in an informed manner between possible interpretations of their input.

In this paper, we address the readability of LS in terms of natural language. To the best of our knowledge, this is the first work that shows how to verbalize LS. As a result, it will help people who are unfamiliar with the underlying technology of LS to understand and interact with it efficiently. The contribution of this paper is twofold. First, we address the readability of LS and propose a generic rule-based approach to produce natural text from LS. Second, we present a selectivity-based approach to generate a summarized verbalization of LS. Our approach is motivated by the pipeline architecture for natural language generation (NLG) systems performed by systems such as those introduced by Reiter & Dale [18].

The rest of this paper is structured as follows: First, we introduce our basic notation in Section 2. Then, we give an overview of our approach underlying LS verbalization in Section 3. We then evaluate our approach with respect to the *adequacy* and *fluency* [5] of the natural language representations it generates in Section 4. After a brief review of related work in Section 5, we conclude our work with some final remarks in Section 6.

Throughout the rest of the paper, we use the following LS shown in Listing 1 as our running example, which is generated by the EAGLE algorithm to link the `ABT-BUY` benchmark dataset from [10], where the source resource x will be linked to the target resource y if our running example's LS holds.

```
OR(jaccard(x.name,y.name)|0.42,trigrams(x.name,y.description)|0.61)
```

Listing 1: Running example.

2 Preliminary

In the following, we present the core of the formalization and notation necessary to implement our LS verbalization. We first give an overview of the grammar that underlies LS. Then, we describe the notation of LS verbalization.

2.1 Link Specification

The link discovery problem is formally defined as follows: Given an input relation ρ (e.g., `owl:sameAs`), a set of source resources S and a set of target resources T, the goal of link discovery is to discover the set $\{(s,t) \in S \times T : \rho(s,t)\}$. Declarative link discovery frameworks define the conditions necessary to generate such links using LS. Several grammars have been used for describing LS in previous work [9,19,15]. In general, these grammars assume that a LS consists of two types of atomic components: *similarity measures* m, which allow the comparison of property values of input resources and *operators* op, which can be used to combine these similarities to more complex specifications. Without loss of generality, we define a similarity measure m as a function $m : S \times T \rightarrow [0, 1]$. We use *mappings* $M \subseteq S \times T$ to store the results of the application of a similarity function to $S \times T$ or subsets thereof. We define a *filter* as a function $f(m, \theta)$. We call a specification *atomic LS* when it consists of exactly one filtering function. A complex specification (*complex LS*) can be obtained by combining two specifications L_1 and L_2 through an *operator* op that allows the results of L_1 and L_2 to be merged. Here, we use the operators $\sqcap$, $\sqcup$ and $\backslash$ as they are complete and frequently used to define LS [19]. A graphical representation of our running example's complex LS from Listing 1 is given in Figure 1.

We define the semantics $[[L]]_M$ of a LS L w.r.t. a mapping M as given in Table 1. Those semantics are similar to those used in languages like SPARQL, i.e., they are defined extensionally through the mappings they generate. The mapping $[[L]]$ of a LS L with respect to $S \times T$ contains the links that will be generated by L. We define the *selectivity score* of a sub-LS $L_s \in L$ as a function $\sigma(L)$ that returns the the F-Measure achieved by the mapping $[[L_s]]$ of L_s by considering the mapping $[[L]]$ generated by the original LS L as its reference mapping.

Fig. 1: Our running example complex LS. The filter nodes are rectangles while the operator nodes are circles.

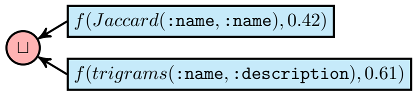
$\sqcup$
$f(Jaccard(\texttt{:name}, \texttt{:name}), 0.42)$
$f(trigrams(\texttt{:name}, \texttt{:description}), 0.61)$

Table 1: Link Specification Syntax and Semantics.

LS	$[[LS]]_M$
$f(m, \theta)$	$\{(s,t) \mid (s,t) \in M \wedge m(s,t) \geq \theta\}$
$L_1 \sqcap L_2$	$\{(s,t) \mid (s,t) \in [[L_1]]_M \wedge (s,t) \in [[L_2]]_M\}$
$L_1 \sqcup L_2$	$\{(s,t) \mid (s,t) \in [[L_1]]_M \vee (s,t) \in [[L_2]]_M\}$
$L_1 \backslash L_2$	$\{(s,t) \mid (s,t) \in [[L_1]]_M \wedge (s,t) \notin [[L_2]]_M\}$

2.2 Link Specification Verbalization

Our definition of realization function ζ relies on the formalization of the LS declared in the previous Section. Let $\mathcal{A}$ be the set of all *atomic LS* that can be combined in a *complex LS L*. Let C^S resp. C^T be two sets of constraints that specify the sets S resp. T. Let $\mathcal{M}$ be a set of similarity measures and $\mathcal{T}$ a set of thresholds. In General, a constraint C is a logical predicate. Constraints in LS could state, for example, the `rdf:type` of the elements of the set they describe, i.e., $C(x) \leftrightarrow x$ `rdf:type someClass`, or the features that each element in the set must have, e.g., $C(x) \leftrightarrow (\exists y : x$ `someProperty` $y)$. Each $s \in S$ must abide by each of the constraints $C_1^S \ldots C_m^S$, while each $t \in T$ must abide by each of the constraints $C_1^T \ldots C_k^T$. We call $z \in \mathcal{A} \cup C^S \cup C^T \cup \mathcal{M} \cup \mathcal{T}$ an *atom*. We define the realization function $\zeta : \mathcal{A} \cup C^S \cup C^T \cup \mathcal{M} \cup \mathcal{T} \rightarrow Language$, where *Language* is our target language. In turn, this realization function ζ maps each atom to a word or sequence of words in our target language. Formally, the goal

Table 2: Dependencies used by LS verbalization.

Dependency	**Explanation**
`amod`	Represents the *adjectival modifier* dependency. For example, `amod(ROSE,WHITE)` stands for `w`hite rose.
`dobj`	Dependency between a verb and its *direct object*. For example, `dobj(EAT,APPLE)` expresses "to eat an/the apple".
`nn`	The *noun compound modifier* is used to modify a head noun by the means of another noun. For instance, `nn(FARMER,JOHN)` stands for `f`armer John.
`poss`	Expresses a possessive dependency between two lexical items. For example, `poss(JOHN,DOG)` express John's dog.
`prep_X`	Stands for the preposition `X`, where `X` can be any preposition, such as `via`, `of`, `in` and `b`etween.
`subj`	Relation between *subject* and verb. For example, `subj(PLAY,JOHN)` expresses John plays.

of this paper: first is to construct the extension of ζ to the entire LS so that all *atoms* z can be mapped to their realization $\zeta(x)$. Second : how these atomic realizations can be combined. For the sake of simplicity, we denote the extension of ζ by the same label ζ. We adopt a rule-based approach to achieve this goal, where the rule extending ζ to the entire LS is expressed in a conjunctive manner. This means that for premises $P_1, \ldots, P_n$ and consequences $K_1, \ldots, K_m$ we write $P_1 \wedge \ldots \wedge P_n \Rightarrow K_1 \wedge \ldots \wedge K_m$. The premises and consequences are clarified by using an extension of the Stanford dependencies[3]. Notably, we build on the constructs explained in Table 2. For example, dependency between a *verb* and its *object* is represented as `dobj`$(verb, object)$.

3 Approach

We have now introduced all ingredients necessary for defining our approaches for LS verbalization and summarization. Our goal is to generate a complete and correct natural language representation of an arbitrary LS. Our approach is motivated by the pipeline architecture for natural language generation (NLG) systems as introduced by Reiter & Dale [18]. The NLG architecture consists of three main stages: *document-planner*, *micro-planner* and *surface realizer*. Since this work is the first step towards the verbalization of LS, our efforts will be focused on *document-planner* (as explained in Section 3.1) with an overview of the tasks carried out in the *micro-planner* (Section 3.2). The *surface realizer* is used to create the output text.

[3] For a complete description of the vocabulary, see `http://nlp.stanford.edu/software/dependencies_manual.pdf`.

3.1 Document-Planner

The *document-planner* consists of the content determination process to create messages and the document structuring process that combines those messages. We focus on document structuring to create independently verbalizable messages from the input LS and to decide on their order and structure. These messages are used for representing information. This part is carried out in the preprocessing and processing steps.

Preprocessing: The goal of the preprocessing step is to extract the central information of LS. This step mainly relies on the *atomic LS* where the necessary information can be extracted. The input for this step is the *atomic LS* while the output is the realization of each individual part of the *atomic LS*. To this end, we break down the *atomic LS* into its individual parts, consisting of properties p (for each *atomic LS* there are two properties - 1. p_s for the resource $s \in S$ and 2. p_t for the resource $t \in T$), threshold θ and similarity measure m. After that, on each part of the *atomic LS* we apply the dependency rule introduced in table 1. We start with the realization of similarity measure m (e.g. *jaccard* as stated in our running example in Listing 1) as follows:

1. ζ(m) $\Rightarrow$ nn(m,similarity)

Now, we can combine ζ(m) and $\zeta(\theta)$.

2. $\zeta(m,\theta)$ = ζ(m) $\wedge$ $\zeta(\theta)$ $\Rightarrow$ prep_of($\zeta(\theta)$,$\zeta(m)$)

Furthermore, if θ equals 1, we replace its value by *"exact match"* and in cases where θ is equal to 0, we replace it by *"complete mismatch"*. Otherwise, we keep the θ value (e.g., in the case of our running example). Regarding the properties p_s and p_t, we move the explanation into the processing step since they play an important role in the construction of a *subject* to be used later in sentence building.

Processing: In this step, we aim to map all *atoms* z into their realization function ζ(z) and to define how these atomic realizations are to be combined. The input for this step is the LS and the output is the verbalization of the LS at hand. Given our formalization of LS in Section 2.1, any LS is a binary tree, where the root of the tree is an operator *op* and each of its two branches are LSs. Therefore, we recursively in-order apply our processing step at the LS tree at hand. As the complete verbalization of an atomic LS mainly depends on the properties p_s and p_t, we here distinguish two cases: a *first case* where p_s and p_t are equal, so we only need to verbalize p_s. In this case the realization function of an atomic LS $a \in \mathcal{A}$ is constructed as follows:

3. ζ(a)$\Rightarrow$subj(have,nn(prep_of($\zeta(p_s)$, ζ(source and target)), ζ(resources))) $\wedge$ dobj(have,$\zeta(m,\theta)$)

The *second case* is where the p_s and p_t are not equal. Here, both properties need to be verbalized as follows:

4. $\zeta(p_s,p_t)\Rightarrow$ $\zeta(p_s)$ $\wedge$ $\zeta(p_t)$

3.2 Micro-Planer

The micro-planner is divided into three processes: *lexicalization, referring expression generation* and *aggregation*. We explain each process in the following.

Lexicalization: Within the lexicalization process we decide what specific words should be used to express the content. In particular, we choose the actual nouns, verbs, adjectives and adverbs to appear in the text from a lexicon. Also, we decide which particular syntactic structures to use, for example, whether to use the phrase `the name of the resource` or **resource's name**.

5. $\zeta(p_s) \Rightarrow$ `prep_of( poss(`ζ`(resource),` p_s`),`ζ`(source))`
6. $\zeta(p_t) \Rightarrow$ `prep_of( poss(`ζ`(resource),` p_t`),`ζ`(target))`
7. ζ`(a)`$\Rightarrow$ `subj(have,`$\zeta(p_s, p_t)$`)` $\wedge$ `dobj(have,`$\zeta(m, \theta)$`)`

Applying *preprocessing* and *processing* steps followed by *Lexicalization* step on our running example from Listing 1 generates the following verbalization: `The name of source and target resources has a 42% of Jaccard similarity or the resource's name of the source and the resource's description of the target have a 61% of Trigrams similarity.` Note that our running example contains both cases.

Referring expression generation: Here we carry out the task of deciding which expressions should be used to refer to entities. Considering the example, `the source and the target have a resource's name and they have a 45% of Jaccard similarity`, `they` is referring to the expression `the source and the target`. However, we avoid such a construction in our verbalization because we aim to generate a simple yet readable text that contains the central information of the LS at hand.

Aggregation: The goal of aggregation in NLG is to avoid duplicating information that has already been presented. In our LS verbalization, we mainly focus on the *subject collapsing*, defined in [4] as the process of *"collecting clauses with common elements and then collapsing the common elements"*. Formally, we define *subject* `subj(`v_i`,` s_i`)` as s_i, *object* `dobj(`v_i`,` o_i`)` as o_i

8. $\zeta(s_1)$`=` $\zeta(s_2)$`=...=`$\zeta(s_n) \Rightarrow$ `subj(`v_1`,`s_1`)` $\wedge$ `dobj(`v_1`, coord(`o_1`,`o_2`,...,`o_n`))`

In the Listing 2, we present a second example LS where grouping is applicable.

```
OR(jaccard(x.name,y.name)|0.42,qgrams(x.name,y.name)|0.61)
```

Listing 2: Grouping example.

The original verbalization of LS from Listing 2 is: `The name of source and target resources has a 42% of Jaccard similarity or the name of source and target resources has a 61% of Qgrams similarity.` And after applying grouping, our verbalization will become more compact as follows: `The name of source and target resources has a 42% of Jaccard similarity or a 61% of Qgrams similarity.`

3.3 Summarization

We propose a sentence-scoring-based LS summarization approach. The basic idea behind our summarization approach is to simplify the original LS tree by, in order, pruning LS sub-trees that achieve the minimum *selectivity score*. i.e., keep the information loss minimum. Given an input LS L_i, our summarization approach first generates an ordered list $\mathbf{L}$ of simplified LSs of L_i, where $\mathbf{L}$ is ordered by the selective score of each of its elements in descending order. This step is carried out by iteratively pruning the sub-tree of L_i with the minimum selectivity score.

In cases where a summarization threshold $\tau \in [0, 1]$ is given, the output of our summarization algorithm will be generated by applying our LS verbalization approach to the LS $L \in \mathbf{L}$ with the highest selectivity score $\sigma(L) \leq \tau$. Otherwise, the output of our summarization approach will be a list of the verbalization of the whole list $\mathbf{L}$.

4 Evaluation

We evaluated our approaches for LS verbalization and summarization in order to elucidate the following questions:

Q_1: Does the LS verbalization help the user to better understand the conditions sufficient to link the resources in comparison to the original LS?
Q_2: How fluent is the generated LS verbalization? i.e., how good is the natural language description of the LS verbalization in terms of comprehensibility and readability?
Q_3: How adequate is the generated LS verbalization? i.e., How well does the verbalization capture the meaning of the underlying LS?
Q_4: How much information do we lose by applying our summarization approach?

4.1 Experimental setup

To answer the first three questions, we conducted a *user study* in order to evaluate our LS verbalization. Therefore, we used our approach to verbalize a set of five LSs automatically generated by the EAGLE algorithm [15] for the benchmark datasets of `Amazon-GP`, `ABT-BUY`, `DBLP-ACM`, and `DBLP-Scholar` from [10]. Our user study consists of four tasks, where each task consists of five multiple choice questions[4]. Altogether, we have a group of 18 participants in our user study from the DICE[5] and AKSW[6] research groups. In the following, we explain each task:

[4] The survey interface can be accessed at `https://umfragen.uni-paderborn.de/index.php/186916?lang=en`
[5] `https://dice.cs.uni-paderborn.de/about/`
[6] `http://aksw.org/About.html`

- *Task 1:* This task consists of five identical sub-tasks. For each we present the survey participant a LS and three pairs of source and target resources represented by their respective concise bounded descriptions (CBD)[7] graph. These pairs are matched together based on the provided LS with different degrees of confidence. To this end, the participant is asked to find the best matched pair, and we measure the response time for each participant.
- *Task 2:* This task also consists of five identical sub-tasks. We again follow the same process in *Task 1* of presenting the participant with the CBDs of matched resources, but this time we give the survey participant the verbalization of the LSs. Again, we record the response time of each participant.
- *Task 3:* Within this task, a survey participant is asked to judge the fluency of the provided verbalization. We follow here the machine translation standard introduced in [5]. Fluency captures how good the natural language description is in terms of comprehensibility and readability according to the following six ratings: (6) Perfectly clear and natural, (5) Sounds a bit artificial, but is clearly comprehensible. (May contain minor grammatical flaws.), (4) Sounds very artificial, but is understandable (although may contain significant grammatical flaws), (3) Barely comprehensible, but can be understood with some effort, (2) Only a loose and incomplete understanding of the meaning can be obtained, and (1) Completely not understandable at all.
- *Task 4:* In this task, we provide a survey participant with a LS and its verbalization. They are then asked to judge the adequacy of the verbalization. Here we follow the machine translation standard from [5]. Adequacy addresses how well the verbalization captures the meaning of the LS, according to the following six ratings: (6) Perfect, (5) Mostly correct, although maybe some expressions don't match the concepts very well, (4) Close, but some information is missing or incorrect, (3) There is significant information missing or incorrect, (2) Natural Language (NL) description and LS are only loosely connected, and (1) NL description and LS are in no conceivable way related.

For answering the last question, we conducted an experiment on the benchmark datasets from [10]. We ran the supervised version of the WOMBAT algorithm to generate an automatic LS for each dataset. We again used [19] to configure WOMBAT. Afterwards, we applied our summarization algorithm to each of the generated LSs. Because of the space limitation, we present only the verbalization of the original LS (the ones generated by WOMBAT) as well as the first summarization of it for the `Amazon-GP` and `DBLP-Scholar` datasets in Table 3. The complete results are available on the project website[8].

[7] `https://www.w3.org/Submission/CBD/`
[8] `https://bit.ly/2XKDpKZ`

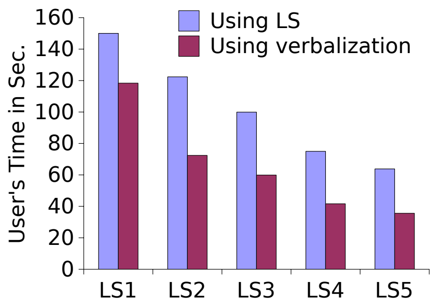

Fig. 2: Average response time of our user study.

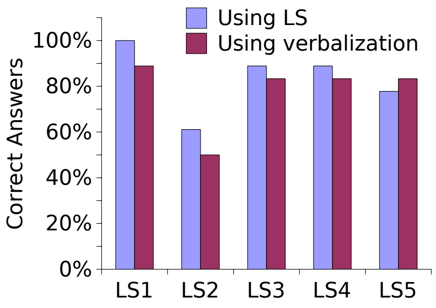

Fig. 3: Correct answers of our user study.

4.2 Results and Discussion

After collecting all the responses of our user study, we filtered out those survey participants who were unlikely to have thoroughly executed the survey (i.e., the ones who took notably less time than the average response time of all other participants) or who were likely distracted while executing it (i.e., the ones who took notably more time than the average time of all other participants). This process reduces the number of valid participants to 16. Our final accepted time window was $3.5-38$ minutes for *Task 1* & *2*. Accordingly, we start our evaluation by comparing the user time required to find the best matched source-target pair using LS (*Task 1*) against using the verbalization of the provided LS (*Task 2*).

As shown in Figure 2, the average user response time with LS verbalization is less than the ones for LS in all the 5 LSs in our users study. On average, using verbalization is 36% faster than using LS. Additionally, we also compared the error rates of participants in *Task 1* & *2*, i.e. the number of incorrect answers per question. As shown in Figure 3, using verbalization we have a higher error rate (5% mean squared error) than when using LS. These results show that using LS verbalization decreases the average response time, which is an indicator that our participants were able to better understand underlying LS using verbalization. Still, using the LS verbalization does not always lead our participants to select the correct answer. This is due to the complexity involved in the underlying LSs, which leads to verbalization that is too long. This answers Q_1. Using our simplification approach on the same LS verbalization leads our participants to achieve better results.

The results of *Task 3* (see Figure 4) show that the majority of the generated verbalizations (i.e., the natural language descriptions) were fluent. In particular, 87% of the cases achieved a rating of 3 or higher. On average, the fluency of the natural language descriptions is 5.2 ± 1.8. This answers Q_2.

For *Task 4*, the average adequacy rating of our verbalization was 5 ± 2.55 (see Figure 5), which we consider to be a positive result. In particular, 40% of all verbalizations were judged to be perfectly adequate and 83% of the cases achieved a rating of 3 or higher. This answers Q_3.

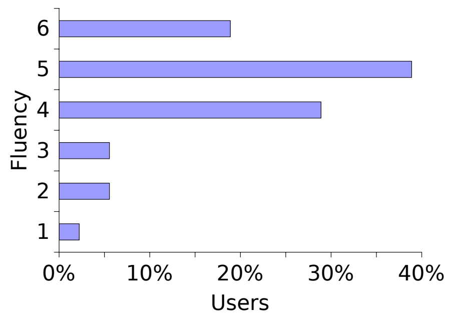

Fig. 4: Fluency results.

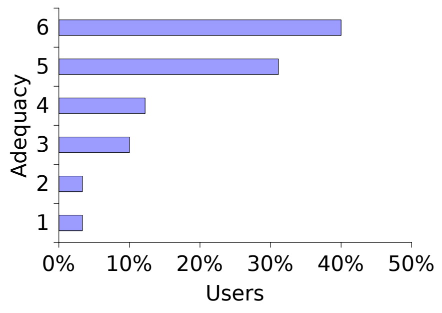

Fig. 5: Adequacy results.

Table 3: Verbalization of different summarization of a LS for the `DBLP-SCHOLAR` and `Amazon-GP` dataset together with respective **F**-measure.

Dataset	**F**	**Verbalization**
`DBLP-SCHOLAR`	1	`The link will be generated if the title of the source and the target resources has a 66% of Cosine similarity or the resource's title of the source and the resource's author of the target has a 43% of Jaccard similarity or the resource's author of the source and the resource's title of the target has a 43% of Trigram similarity`
`DBLP-SCHOLAR`	0.88	`The link will be generated if the title of the source and the target resources has a 66% of Cosine similarity`
`Amazon-GP`	1	`The link will be generated if the resource's title of the source and the resource's name of the target has a 48% of Cosine similarity or the description of the source and the target resources has a 43% of Cosine similarity or the resource's title of the source and the resource's description of the target has a 43% of Jaccard similarity`
`Amazon-GP`	0.97	`The link will be generated if the resource's title of the source and the resource's name of the target has a 48% of Cosine similarity`

5 Related work

While we believe that this is the first work that shows how to verbalize LS, related work comes from three research areas: declarative link discovery approaches, verbalization of Semantic data and text summarization.

Declarative Link Discovery frameworks rely on complex LS to express the conditions necessary for linking resources within RDF datasets. For instance, state-of-the-art LD frameworks such as LIMES [13] and SILK [9] adopt a property-based computation of links between entities. All such frameworks enable their users to manually write LS and excute it against source-target resources. In recent years, the problem of using machine learning for the automatic generation of accurate LS has been addressed by most of the link discovery frameworks. For example, the SILK framework [9] implements a batch learning approach for the discovery LS, based on genetic programming, which is similar to the approach presented in [3]. For the LIMES framework, the RAVEN algorithm [14] is an active learning approach that treats the discovery of specifications as a classification problem. In RAVEN, the discovery of LS is done by first finding class and property mappings between knowledge bases automatically. It then uses these mappings to compute linear and boolean classifiers that can be used as LS. Eagle [15] has addressed the readability of LS alongside accuracy and efficiency. However,the generated LS is still expressed in a declarative manner. Recently, the Wombat algorithm [19] has implemented a machine leaning algorithm for automatic LS finding by using generalization via an upward refinement operator.

With the recent demand on new explainable machine learning approaches, comes the need for *the verbalization of semantic data* involved within such approaches. For example, [17] expands on an approach for converting RDF triples into Polish. The authors of [12] espouse a reliance on the Linked Data Web being created by reversing engineered structured data into natural language. In their work [20], the same authors show how this approach can be used to produce text out of RDF triples. Yet another work, [11], generated natural language out of RDF by depending on the BOA framework [8,7] to compute the trustworthiness of RDF triples using the Web as background knowledge. Other approaches and concepts for verbalizing RDF include [16] and [22]. Moreover, approaches to verbalizing first-order logics [6] are currently being devised. In very recent work [21], the authors have addressed the limitations of adapting rule-based approaches to generate text from semantic data by proposing a statistical model for NLG using neural networks.

The second fold of our approach is the summarization of LS, which is related to work in the area of *text summarization* with a focus on sentence scoring techniques. The work [1] surveys many sentence scoring techniques. Furthermore, the survey [2] addresses many text summarization methods. However, in our summarization technique the summarization score is user-defined.

6 Conclusions and Future Work

In this paper, we presented LSVS, an approach for verbalizing LS. LSVS produces both a direct literal verbalization of the content of the LS and a more natural aggregated version of the same content. We presented the key steps of our approach and evaluated it with a user study. Our evaluation shows that the verbalization generated by our approach is both complete and easily understandable. Our approach not only accelerates the understanding of LS by expert users, but also enables non-expert users to understand the content of LS. Still, our evaluation shows that the fluency of our approach is worse when the LS gets more complex and contains different operators. In future work, we will thus improve upon our aggregation to further increase this fluency. Moreover, we will devise a consistency checking algorithm to improve the correctness of the natural language generated by our approach.

References

1. Assessing sentence scoring techniques for extractive text summarization. *Expert Systems with Applications*, 2013.

2. M. Allahyari, S. A. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut. Text summarization techniques: A brief survey. *CoRR*, 2017.
3. M. G. Carvalho, A. H. F. Laender, M. A. Gonçalves, and A. S. da Silva. Replica identification using genetic programming. ACM, 2008.
4. H. Dalianis and E. Hovy. Aggregation in natural language generation. Springer, 1996.
5. G. Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of HLT*, pages 138–145, 2002.
6. N. E. Fuchs. First-order reasoning for attempto controlled english. 2010.
7. D. Gerber and A.-C. N. Ngomo. Extracting multilingual natural-language patterns for rdf predicates. In *EKAW*, pages 87–96, 2012.
8. D. Gerber and A.-C. Ngonga Ngomo. Bootstrapping the linked data web. In *1st Workshop on Web Scale Knowledge Extraction @ ISWC 2011*, 2011.
9. R. Isele, A. Jentzsch, and C. Bizer. Efficient Multidimensional Blocking for Link Discovery without losing Recall. In *WebDB*, 2011.
10. H. Köpcke, A. Thor, and E. Rahm. Comparative evaluation of entity resolution approaches with fever. *Proc. VLDB Endow.*, 2(2):1574–1577, 2009.
11. J. Lehmann, D. Gerber, M. Morsey, and A.-C. Ngonga Ngomo. Defacto - deep fact validation. In *ISWC*, 2012.
12. C. Mellish and X. Sun. The semantic web as a linguistic resource: opportunities for natural language generation. 2006.
13. A.-C. N. Ngomo and S. Auer. Limes - a time-efficient approach for large-scale link discovery on the web of data. In *IJCAI*, 2011.
14. A.-C. Ngonga Ngomo, J. Lehmann, S. Auer, and K. Höffner. Raven – active learning of link specifications. In *Proceedings of OM@ISWC*, 2011.
15. A.-C. Ngonga Ngomo and K. Lyko. Eagle: Efficient active learning of link specifications using genetic programming. Springer Berlin Heidelberg, 2012.
16. H. Piccinini, M. A. Casanova, A. L. Furtado, and B. P. Nunes. Verbalization of rdf triples with applications. In *ISWC - Outrageous Ideas track*, 2011.
17. A. Pohl. The polish interface for linked open data. In *Proceedings of the ISWC 2010 Posters & Demonstrations Track*, pages 165–168, 2011.
18. E. Reiter and R. Dale. *Building natural language generation systems.* Cambridge University Press, New York, NY, USA, 2000.
19. M. Sherif, A.-C. Ngonga Ngomo, and J. Lehmann. WOMBAT - A Generalization Approach for Automatic Link Discovery. Springer, 2017.
20. X. Sun and C. Mellish. An experiment on "free generation" from single rdf triples. Association for Computational Linguistics, 2007.
21. P. Vougiouklis, H. Elsahar, L.-A. Kaffee, C. Gravier, F. Laforest, J. Hare, and E. Simperl. Neural wikipedian: Generating textual summaries from knowledge base triples. *Journal of Web Semantics*, 52-53:1 – 15, 2018.
22. G. Wilcock and K. Jokinen. Generating Responses and Explanations from RDF/XML and DAML+OIL, 2003.