RDFPYREALB at the GEM'24 Data-to-text Task: Symbolic English Text Generation from RDF Triples

Guy Lapalme
RALI-DIRO / Université de Montréal
CP. 6128, Succ. Centre-Ville
Montréal, Québec, Canada
lapalme@iro.umontreal.ca

Abstract

We present a symbolic system, written in Python, used to participate in the English Data-to-text generation task of the *GEM Shared Task at the Generation Challenges (INLG'24)*. The system runs quickly on a standard laptop, making it fast and predictable. It is also quite easy to adapt to a new domain.

1 Introduction

This paper describes PYREALB, a system for tackling the Data-to-text generation task of the *GEM Shared Task at the Generation Challenges (INLG'24)* (Mille et al., 2024). It uses a symbolic approach to this problem, which has become almost *forgotten* due to the popularity of neural networks and large language models. We thought it would be interesting to compare the results between computationally intensive methods that can sometimes be difficult to control with a *predictable*, lightweight and fast symbolic approach.

The system is conceptually simple, each RDF triple corresponds to a sentence in which the subject and the object of a triple are mapped almost verbatim as subject and object of the sentence. The predicate of the triple corresponds to a verb phrase that determines the structure of the sentence. The system orders predicates to create a meaningful story, and merges parts of sentences when they have shared subjects or predicates. The final realization is performed using PYREALB, a French-English realizer used in some data to text applications (Lapalme, 2023).

PYREALB derived from our submission to the WebNLG Challenge (Lapalme, 2020) 2020 in which the text realization was performed through an API that sent JSON structures to a JSREALB[1] server that returned the final text. In this case, we perform the realization directly in Python. Our paper provided a critical review of the data and discussed the suitability of this competition results in a wider Natural Language Generation setting. These remarks are still valid, given that the data for this shared task is the same or a textual replacement of entities without changing the organization of the RDF triples. Provisions have been made to remove singleton sets from the evaluation in this competition, thus making sentence realization a bit more challenging.

2 Text Generation

We recall that an RDF triple is composed of three URIs. In this dataset, they are replaced by English tokens, corresponding to the subject, the predicate and the object. An object can also be a constant string, a date or a number. The predicate of a triple declares a relation between the subject and the object, such as `Campeonato_Brasileiro_Série_C | country | Brazil`, in which `Campeonato_Brasileiro_Série_C` is the subject (a Brazilian Soccer competition), `country` the predicate indicating that the subject *takes place in the country* indicated by the object `Brazil`.

To illustrate our NLG process, we use the set of triples shown in Table 1 with the corresponding generated English sentence.

The first step in text generation is to determine what information to include in the text. In the context of this shared task, this is given: it consists of at most 7 triples, and only 4% of the sets are made up of seven triples. Moreover, 20% of the triples are singletons that are easy to generate, but they were not submitted for human evaluation. Since the predicate of a triple indicates a relationship between its subject and object, in our case, it maps to a verb that links the subject and object of the sentence realizing this triple.

[1] https://github.com/rali-udem/jsRealB

Proceedings of the 17th International Natural Language Generation Conference: Generation Challenges, pages 54–58
September 23 –27, 2024.

```
<entry category="SportsTeam" eid="Id649" shape="(X (X) (X (X)) (X (X) (X)))" shape_type="mixed" size="7"
    >
    <modifiedtripleset>
        <mtriple>Estádio_Municipal_Coaracy_da_Mata_Fonseca | location | Arapiraca</mtriple>
        <mtriple>Agremiacao_Sportiva_Arapiraquense | league | Campeonato_Brasileiro_Série_C</mtriple>
        <mtriple>Campeonato_Brasileiro_Série_C | champions | Vila_Nova_Futebol_Clube</mtriple>
        <mtriple>Campeonato_Brasileiro_Série_C | country | Brazil</mtriple>
        <mtriple>Agremiacao_Sportiva_Arapiraquense | numberOfMembers | 17000</mtriple>
        <mtriple>Agremiacao_Sportiva_Arapiraquense | ground | Estádio_Municipal_Coaracy_da_Mata_Fonseca</
            mtriple>
        <mtriple>Agremiacao_Sportiva_Arapiraquense | manager | Vica</mtriple>
    </modifiedtripleset>
</entry>
```

Agremiacao Sportiva Arapiraquense has Vica as manager, it has 17,000 members and plays in the Campeonato Brasileiro Série C league. It plays in Estádio Municipal Coaracy da Mata Fonseca located inside Arapiraca. Campeonato Brasileiro Série C is from Brazil and where Vila Nova Futebol Clube were champions.

Table 1: The top part shows a triple set from `D2T-1-FA-WebNLG_Factual.xml`, the content of the `originaltripleset` is not shown here because it is ignored in the competition. The bottom part shows the realized sentence produced by RDFPYREALB from this input.

2.1 Microplanning

Since triples are unordered, the first critical step is organizing them to create an *interesting story*. First, we group the triples based on their subjects. We then sort the triples within each group. For example, when describing a person, we can begin with their date and place of birth, then move on to their activities, before finishing with their retirement and death. For a university or a football club, we would start with its creation date, then its directors and finally its activities. To achieve this ordering, each predicate is assigned a *priority* that is used to sort the triples. These priorities were established by hand and are currently independent of the category of the subject.

Then the groups are processed in descending order of triplets. We also query DBpedia to determine whether the category of a group subject corresponds to the specified category in the data. If so, we increase its score so that the text begins with this subject. Each group forms a sentence as a coordination of *subsentences*. Because long coordinated sentences are often difficult to follow, groups of more than three triplets are split into two sentences. In order to avoid very short sentences, a group with a single triple is combined using a subordinate when its subject is the object of another triple in a bigger group. Table 1 shows an example of this in which the last triple of the first group has been combined with the last triple.

Table 2 shows the result of the sorting and grouping process on the example of Table 1. The four triples having `Agremiacao_Sportiva_Arapiraquense` as subject are grouped and sorted to form a coherent story. This input is used for realizing the three sentences shown in the bottom part of Table 1 using PYREALB.

```
Agremiacao_Sportiva_Arapiraquense
    manager Vica;
    numberOfMembers 17000;
    league Campeonato_..._C;
    ground Estádio_..._Fonseca.
Campeonato_..._C
   country Brazil;
   champions Vila_Nova_Futebol_Clube.
Estádio_..._Fonseca
    location Arapiraca.
```

Table 2: `mtriples` from Table 1 sorted and grouped, shown as a Turtle-like formalism, used as input for RDFPYREALB. Predicates and objects sharing the same subject are shown indented and separated by semicolons. Some tokens are shown here with ellipsis to make them fit in the two-column format. The bottom of Table 1 corresponds closely to this *text plan*.

2.2 Surface realization

For the final realization step, we use PYREALB a Python implementation of JSREALB (Lapalme, 2022) in which programming language instructions create data structures corresponding to the constituents of the sentence to be produced. Once the data structure is built, it is traversed to produce the list of words in the sentence, taking care of issues such as conjugation, agreement, capitalization, and other *small* details that help readers and evaluators.

The data structure is built by calls to functions whose names were chosen to be similar to the symbols typically used for constituent syntax trees, such as a *Terminal* (e.g. `N` (Noun), `V` (Verb), `A` (adjective), `D` (determiner), `Q` which quotes its parameter thus allowing *canned text*) or a *Phrase* (e.g. `S` (Sentence), `NP` (Noun Phrase), `VP` (Verb Phrase)).

```
S(Pro("I").g('n'),
  VP(V("play"),
     PP(P("in"),
        SP(Q("Estádio␣Fonseca"),
           VP(V("locate").t('pp'),
              PP(P("inside"),
                 Q("Arapiraca")))))))
```

Table 3: Top: Python functional notation for a PYREALB expression realized as: It plays in Estádio Fonseca located inside Arapiraca

Features added to structures with the dot notation can modify their properties. Terminals can specify their person, number and gender. Phrases can have a negation or be put into passive mode. A noun phrase can be pronominalized, and coordinated phrases are automatically processed, inserting appropriate commas and conjunctions between coordinated elements. Table 3, shows the Python calls to create an internal structure that is realized as an English sentence.

2.3 Sentence Templates

The goal is to transform the structure of Table 2 into that of Table 3. We have manually defined 250 templates corresponding to the most frequent predicates in the set (those with 10 or more occurrences). When no defined template can be found, we use a default template (described in Section 2.4), which was used in 5% of cases.

A predicate `p` corresponds to a Python lambda expression whose parameter is the object `o`. The predicate is called to create a sentence with the subject `s`. The actual parameters are quoted strings of the subject or object of the triple, but replacing underscores by spaces with special cases for numbers and dates.

For example, given the two following Python definitions:

```
managerP = lambda o: VP(V("have"),
                        o,
                        Adv("as"),
                        N("manager"))
sentence = lambda s,p,o: S(Q(s),
                           p(Q(o)))
```

the call

```
sentence("Agremiacao",
         managerP,"Vica")
```

creates the following structure:

```
S(Q("Agremiacao"),
  VP(V("have"),
     Q("Vica"),
     Adv("as"),
     N("manager")))
```

which is verbalized as Agremiacao has Vica as manager. by PYREALB. This is the basic mechanism for creating sentence structures that can be combined in various ways.

```
"city": (30, False, [
    lambda o:VP(V("be"), _from(o)),
    lambda o:VP(_vpas("locate"), _in(o))]),
"country": (40, False, "city"),
"ground": (50, True, [
    lambda o:VP(V("play"), _in(o))]),
"league": (50, True, [
    lambda o:VP(V(oneOf("be", "play", "compete")),
        _in(NP(D("the"), o, N("league"))))]),
"manager": (20, False, [
    lambda o:VP(_vpas("manage"), _by(o)),
    lambda o:VP(V("have"),o,
                Adv("as"),N("manager"))]),
```

Table 4: A few Python templates using auxiliary function to build passive verbs (`_vpas`) or prepositional phrases such as `_from(..)` or `_in(..)`

Templates are organized in a dictionary (see Table 4). The name of the predicate is the key, and the value is a 3-tuple with the following elements: a priority (a number between 0 and 100) used for sorting, a boolean indicating if its subject can be a human, and a list of lambda expressions that can verbalize this predicate, one of which is randomly chosen at the realization time.

Templates associated with predicates were developed by looking at `lex` elements in the original WebNLG training corpus. When two templates have the same realizations, the third element of the pair is the name of the original predicate (see `country` in Table 4).

Once we agreed on this template structure, writing them became relatively easy. It takes less than minute to write a lambda defining a constituent expression to reproduce some of them. We noticed that many lexicalizations are often very similar; crowd workers seem to often rely on copy-pasting the subject and the object.

Unfortunately, the names of the predicates used in the Wikidata dataset were different for the same relation. So we developed a mapping between them, as shown in Table 5.

2.4 Default Template

When a predicate is not in the table, a default template is created. By detecting case changes, the name of the predicate is split into *words* and taken as the subject of the be auxiliary, the object is used as an attribute. For example,

```
servedAsChiefOfTheAstronautOfficeIn =>
   Q("served␣as␣chief␣of␣the␣astronaut␣
       office")
```

In the final sentence, the subject of the triple is taken as subject of the be auxiliary, the object of

```
wikidata_properties = {
    'Occupation':"occupation",
    'PlaceOfBirth':"birthPlace",
    'DateOfBirth':"birthDate",
    'PositionHeld':"position",
    'HasChild':"have_as_child",
    'PlaceOfDeath':"deathPlace",
    'Spouse':"spouse",
    'ParticipantIn':"competeIn",
    'HasFather':"have_as_father",
    ...
}
```

Table 5: Mapping between the names of predicates used in the Wikidata dataset used as key and the name used in the WebNLG dataset. When a name contains an underscore (e.g., `have_as`), then a custom verb phrase pattern is used.

the triple is used as an attribute. For example, the triple

```
Alan_Shepard |
    servedAsChiefOfTheAstronautOfficeIn
     | 1963
```

is realized as Alan Shepard served as chief of the astronaut office in is 1963. which is not colloquial but understandable.

2.5 Text aggregation

In some cases, dealing with related information (e.g., birth date and place), combining templates using only their complements (i.e., their last element) will simplify the text. For this we define groups of predicates that can be combined at realization time. When two or three triples are merged into a single sentence, the subject is used at the start but a pronoun is used for the following references. Currently, a very simple system is used for choosing the pronoun: if the predicate is coded as being applicable to a human and the gender of the subject obtained by querying DBPedia is `male`, he is used, if it is `female` then she is chosen, otherwise it is used. When a single triple whose subject is used as object of another, it is combined with the subordinate using a pronoun: who if the predicate applies to a human, otherwise that.

3 Running the System

The PYREALB is publicly available, its source code[2] is licensed under Apache-2.0 and the linguistic resources are licensed under CC-BY-SA-4.0. It can also be used as a PyPi module.[3]

The Python code for RDFPYREALB is a demo[4] of PYREALB. The demo, launched with `WebGenerate.py`, is illustrated with English and French texts realized from 6 and 7 triples selected from the original WebNLG 2020 data, which give rise to the most interesting and challenging texts. The script for realizing the submissions to this shared task is `GEM-2024.py`.

4 Comments on the task data

#sbj	WN-trn	WN-dev	WN-test	WkData
1	57%	60%	74%	88%
2	33%	31%	20%	9%
3	9%	8%	5%	2%
4	<1%	<1%	-	<1%
#tpl	13 124	1 667	1 779	1 712

Table 6: percentages of the number of subjects in different triple sets (WN- is WebNLG-2020, trn, dev and test). WN-tst WebNLG-based (D2T-1) and WkData is the Wikidata-based (D2T-2) of this competition.

In a previous paper (Lapalme, 2020), we argued that the *simplified triple* format of WebNLG does not adequately represent the problem of realizing semantic web data. It short-circuits many important issues, such as the lexical selection of the subject and object. Additionally, the relation names do not conform to the well-established W3C naming conventions. We now raise another issue that we did not notice at the time: the number of distinct subjects in triple sets. Table 6 shows the distribution of the number of subjects in the *factual* sets of data; since the both *fictional* and the *counterfactual* were derived from the factual, their distribution is the same. We see that the vast majority of triples have a single subject: 74% for WebNLG and 80% for WikiData. This simplifies greatly the problem of the text organization leaving only the problem of splitting into one or two sentences.

5 Conclusion

This paper described a symbolic approach to tackling the GEM 2024 SHARED TASK. The approach relies on PYREALB, an existing text realizer that takes care of most of the low-level grammatical aspects, so the pattern could be specified at a relatively high level. After a few false starts and once the overall program organization was settled, it was relatively easy for me to develop and organize the patterns. The preliminary automated scores seems quite competitive compared to those of the other

[2] https://github.com/lapalme/pyrealb
[3] https://pypi.org/project/pyrealb/
[4] https://github.com/lapalme/pyrealb/tree/main/demos/RDFpyrealb

participants, whom we conjecture mostly used machine learning approaches. In fact, almost all systems seem to obtain quite similar results depending on the scoring method. RDFPYREALB is very fast and can easily be adapted to new domains. Considering that adding one new predicate takes about one minute, developing 250 new ones would take about four hours. Machine learning could be used to develop new templates, although we doubt that it would be any faster.

References

Guy Lapalme. 2020. RDFjsRealB: a symbolic approach for generating text from RDF triples. In *WebNLG 2020 : 3rd Workshop on Natural Language Generation from the Semantic Web*, pages 144–153, Dublin, Ireland (virtual). SIGGEN.

Guy Lapalme. 2022. The jsRealB text realizer: Organization and use cases. (arXiv:2012.15425).

Guy Lapalme. 2023. Data-to-text bilingual generation. Technical Report arXiv:2311.14808.

Simon Mille, João Sedoc, Yixin Liu, Elizabeth Clark, Agnes Axelsson, Miruna-Adriana Clinciu, Yufang Hou, Saad Mahamood, Ishmael Obonyo, and Lining Zhang. 2024. The 2024 GEM shared task on multilingual data-to-text generation and summarization: Overview and preliminary results. In *Proceedings of the 17th International Conference on Natural Language Generation: Generation Challenges*, Tokyo, Japan. Association for Computational Linguistics.