Natural Language Generation for Non-Expert Users

Abstract

Motivated by the difficulty in presenting computational results, especially when the results are a collection of atoms in a logical language, to users, who are not proficient in computer programming and/or the logical representation of the results, we propose a system for automatic generation of natural language descriptions for applications targeting mainstream users. Differently from many earlier systems with the same aim, the proposed system does not employ templates for the generation task. It assumes that there exist some natural language sentences in the application domain and uses this repository for the natural language description. It does not require, however, a large corpus as it is often required in machine learning approaches. The systems consist of two main components. The first one aims at analyzing the sentences and constructs a Grammatical Framework (GF) for given sentences and is implemented using the Stanford parser and an answer set program. The second component is for sentence construction and relies on GF Library. The paper includes two use cases to demostrate the capability of the system. As the sentence construction is done via GF, the paper includes a use case evaluation showing that the proposed system could also be utilized in addressing a challenge to create an abstract Wikipedia, which is recently discussed in the BlueSky session of the 2018 International Semantic Web Conference.

Introduction

Natural language generation (NLG) has been one of the key topics of research in natural language processing, which was highlighted by the huge body of work on NLG surveyed in BIBREF0, BIBREF1. With the advances of several devices capable of understanding spoken language and conducting conversation with human (e.g., Google Home, Amazon Echo) and the shrinking gap created by the digital devices, it is not difficult to foresee that the market and application areas of NLG systems will continue to grow, especially in applications whose users are non-experts. In such application, a user often asks for certain information and waits for the answer and a NLG module would return the answer in spoken language instead of text such as in question-answering systems or recommendation systems. The NLG system in these two applications uses templates to generate the answers in natural language for the users. A more advanced NLG system in this direction is described in BIBREF2, which works with ontologies annotated using the Attempto language and can generate a natural language description for workflows created by the systems built in the Phylotastic project. The applications targeted by these systems are significantly different from NLG systems, whose main purpose is to generate high-quality natural language description of objects or reports, such as those reported in the recent AAAI conference BIBREF3, BIBREF4, BIBREF5.

The present paper is motivated by the need to generate natural language description of computational results to non-expert users such as those developed in the Phylotastic project. In this project, the users are experts in evolutionary biology but are none experts in ontologies and web services. When a user places a request, he/she will receive a workflow consisting of web services, whose inputs and outputs are specified by instances of classes in the ontologies working with web services, as well as the ordering and relationships between the services. To assist the user in understanding the workflow, a natural language description of the workflow is generated. In order to accomplish the task, the NLG system in the Phylotastic project proposes to annotate elements of the ontologies using Attempto, a simple subset of English with precisely defined syntax and semantics.

In this paper, we propose a system that addresses the limitation of the system discussed in the Phylotastic project BIBREF2. Specifically, we assume that the annotations given in an ontology are natural language sentences. This is a reasonable assumption given that the developers of an ontology are usually those who have intimate knowledge about entities described in the ontology and often have some sort of comments about classes, objects, and instances of the ontology. We then show that the system is very flexible and can be used for the same purpose with new ontologies.

The rest of the paper is organized as follows. Section SECREF2 briefly reviews the basics of Grammatical Framework (GF)BIBREF6. Section SECREF3 describes the main modules of the system. Section SECREF4 includes two use cases of the system using an available ontologies against in the context of reasoning about ontologies. Specifically, it compares with the system used in the Phylotastic project and an ontology about people. This section also contains a use case that highlights the versatility of the proposed system by addressing a challenge to create an abstract Wikipedia BIBREF7. Related works are discussed in Section SECREF5. Section SECREF6 concludes the paper.

Background: Grammatical Framework

The Grammatical Framework (GF) BIBREF6 is a system used for working with grammars. The GF Resource Grammar Library (RGL) covering syntax of various languages is the standard library for GF. A GF program has two main parts. The first part is the Abstract syntax which defines what meanings can be expressed by a grammar. The abstract syntax defines categories (i.e., types of meaning) and functions (i.e., meaning-building components). An example of an abstract syntax:

Here, Message, People, Action and Entity are types of meanings. startcat flag states that Message is the default start category for parsing and generation. simple_sent is a function accepting 3 parameters, of type People, Action, Entity. This function returns a meaning of Message category. Intuitively, each function in the abstract syntax represents a rule in a grammar. The combination of rules used to construct a meaning type can be seen as a syntax tree.

The second part is composed of one or more concrete syntax specifications. Each concrete syntax defines the representation of meanings in each output language. For example, to demostrate the idea that one meaning can be represented by different concrete syntaxes, we create two concrete syntaxes for two different languages: English and Italian. To translate a sentence to different languages, we only need to provide the strings representing each word in corresponding languages. The GF libraries will take responsibility to concatenate the provided strings according to the language grammar to create a complete sentence, which is the representations of the meaning, in the targeted language. The corresponding concrete syntaxes that map functions in the abstract grammar above to strings in English and in Italian is:

In these concrete syntaxes, the linearization type definition (lincat) states that Message, People, Action and Entity are type Cl (clause), NP (noun phrase), V2 (two-place verb), and NP respectively. Linearization definitions (lin) indicate what strings are assigned to each of the meanings defined in the abstract syntax. To reduce same string declaration, the operator (oper) section defines some placeholders for strings that can be used in linearization. The mkNP, mkN, mkV2, etc. are standard constructors from ConstructorsEng/Jpn library which returns an object of the type NP, N or V2 respectively.

GF has been used in a variety of applications, such as query-answering systems, voice communication, language learning, text analysis and translation, natural language generation BIBREF8, BIBREF9, automatic translation.

The translation from English to Italian can be performed as follows in the GF API:

The above command line produces a syntax tree of the sentence “Bill plays soccer” then turn that tree into a PeopleIta sentence (in Italian) which is displayed in the second line. Figure FIGREF6 shows the meaning in the abstract syntax is represented in Japanese and in Italian, i.e. the two strings represent the same meaning.

Method

To generate a sentence, we need a sentence structure and vocabularies. Our system is developed to emulate the process of a person learning a new language and has to make guesses to understand new sentences from time to time. For example, someone, who understands the sentence “Bill plays a game” would not fully understand the sentence “Bill plays a popular board game” without knowing the meaning of “popular” and “board game” but could infer that the latter sentence indicates that its subject plays a type of game.

The overall design of our system is given in Figure FIGREF7. Given a paragraph, our system produces a GF program (a pair of an abstract and a concrete syntax), which can be used for sentence generation. The system consists of two components, understanding sentences and generating GF grammar. The first component is divided into two sub-components, one for recognizing the sentence structure and one for recognizing the sentence components. The second component consists of a GF grammar encoder and a GF grammar exporter. The encoder is responsible for generating a GF grammar for each sentence, while the exporter aggregates the grammars generated from the encoder, and produces a comprehensive grammar for the whole paragraph.

Method ::: Sentence Structure Recognition

The sentence structure recognition process involves 2 modules: natural language processing (NLP) module and logical reasoning on result from NLP module. In this paper, we make use of the Stanford Parser tools described in BIBREF10, BIBREF11, BIBREF12, BIBREF13, BIBREF14

The NLP module tokenizes the input free text to produce a dependency-based parse tree and part-of-speech tag (POS tag). The dependency-based parse tree and the POS tag are then transform into an answer set program (ASP) BIBREF15 which contains only facts. Table TABREF13 shows the transformation of the result of NLP module into an ASP program for the sentence “Bill plays a game”. In this table, nsubj, det, dobj and punct denote relations in the dependency-based parse tree, and mean nominal subject, determiner, direct object and punctuation respectively. Full description of all relations in a dependency-based parse tree can be found in the Universal Dependency website. The second set of notations are the POS tag PRP, VBP, DT and NN corresponding to pronoun, verb, determiner and noun. Readers can find the full list of POS tag in Penn Treebank Project.

From the collection of the dependency atoms from the dependency-based parse tree, we determine the structure of a sentence using an ASP program, called $\Pi _1$ (Listing).

Each of the rule above can be read as if the right-hand side is true then the left-hand side must be true. These rules define five possible structures of a sentence represented by the atom structure(x,y). x and y in the atom structure(x,y) denote the type of the structure and the number of dependency relations applied to activate the rule generating this atom, respectively. We refer to y as the i-value of the structure. For example, $structure(1,1)$ will be recognized if the nsubj relation is in the dependency-based parse tree; $structure(3,3)$ needs 3 dependency relations to be actived: nsubj, xcomp and dobj. We often use structure #x to indicate a structure of type x.

Together with the collection of the atoms encoding the relations in the dependency-based parse tree, $\Pi _1$ generates several atoms of the form $structure(x,y)$ for a sentence. Among all these atoms, an atom with the highest i-value represents the structure constructed using the highest number of dependency relations. And hence, that structure is the most informative structure that is recoginized for the sentence. Observe that $structure(1,1)$ is the most simplified structure of any sentence.

Method ::: Sentence Components Recognition

The goal of this step is to identify the relationship between elements of a sentence structure and chunks of words in a sentence from the POS tags and the dependency-based parse tree. For example, the sentence “Bill plays a game” is encoded by a structure #2 and we expect that Bill, plays, and game correspond to the subject, verb, and object, respectively.

We begin with recognizing the main words (components) that play the most important roles in the sentence based on a given sentence structure. This is achieved by program $\Pi _2$ (Listing). The first four rules of $\Pi _2$ determine the main subject and verb of the sentence whose structure is #1, #2, #3, or #5. Structure #4 requires a special treatment since the components following tobe can be of different forms. For instance, in “Cathy is gorgeous,” the part after tobe is an adjective, but in “Cathy is a beautiful girl,” the part after tobe is a noun, though, with adjective beautiful. This is done using the four last rules of $\Pi _2$.

The result of program $\Pi _2$ is an one-to-one mapping of some of the words in the sentence into the importaint components of a sentence, called main components, i.e. subject, object and verb. The mapping is constructed by using the core arguments in Universal Dependency Relations . Since not every word in the sentence is in a core argument relation, there are some words in the sentence that are not in the domain of the mapping that $\Pi _2$ produces. We denote these words are complement components. To identify these words, we encode the Non-core dependents and Nominal dependents from Universal Dependency Relations into the set of rules in program $\Pi _3$.

Program $\Pi _3$ (Listing), together with the atoms extracted from the dependency-based parse tree such as $compound(P,N)$ (N is compound noun at the position P in the sentence), $amod(P,J)$ (J is an adjective modifier), etc., is used to identify the complement components of the main components computed by $\Pi _2$ while maintaining the structure of the sentence created by $\Pi _1$. For example, a complement of a noun could be another noun (as “board” in “board game”), or an adjective (as “popular” in “popular board game”), or a preposition (as “for adults” in “board game for adults”).

The input of Program $\Pi _3$ is the position (pos) of the word in the sentence. Program $\Pi _3$ is called whenever there is a new complement component discovered. That way of recursive calls is to identify the maximal chunk of the words that support the main components of the sentence. The result of this module is a list of vocabularies for the next steps.

Method ::: GF Grammar Encoder

The goal of the encoder is to identify appropriate GF rules for the construction of a GF grammar of a sentence given its structure and its components identified in the previous two modules. This is necessary since a sentence can be encoded in GF by more than one set of rules; for example, the sentence “Bill wants to play a game” can be encoded by the rules

Bill $\rightarrow $ NP, want $\rightarrow $ VV, play $\rightarrow $ V2, game $\rightarrow $ NP and one of the sets of GF rules in the table below:

In GF, NP, VV, V2, VP, and Cl stand for noun phrase, verb-phrase-complement verb, two-place verb, verb phrase and clause, respectively. Note that although the set of GF grammatical rules can be used to construct a constituency-based parse tree , the reverse direction is not always true. To the best of our knowledge, there exists no algorithm for converting a constituency-based parse tree to a set GF grammar rules. We therefore need to identify the GF rules for each sentence structure.

In our system, a GF rule is assigned to a structure initially (Table TABREF19). Each rule in Table TABREF19 represents the first level of the constituency-based parse tree. It acts as the coordinator for all other succeeding rules.

Given the seed components identified in Section SECREF15 and the above GF rules, a GF grammar for each sentence can be constructed. However, this grammar can only be used to generate fairly simple sentences. For example, for the sentence “Bill plays a popular board game with his close friends.”, a GF grammar for structure #2 can be constructed, which can only generate the sentence “Bill plays game.” because it does not contain any complement components identified in Section SECREF15. Therefore, we assgin a set of GF rules for the construction of each parameter in the GF rules in Table TABREF19. The set of GF rules has to follow two conventions. The first one is after applying the set of rules to some components of the sentence, the type of the production is one of the type in Table TABREF19, e.g. NP, VP, Cl, $V2$, ... The second convention is that the GF encoder will select the rules as the order from top to bottom in Table TABREF20. Note that the encoder always has information of what type of input and output for the rule it is looking for.

For instance, we have “game” is the object (main components), and we know that we have to construct “game” in the result GF grammar to be a NP (noun phrase). Program $\Pi _2$ identifies that there are two complement components for the word “game”, which are “board” and “popular”, a noun and an adjective respectively. The GF encoder then select the set of rules: N $\rightarrow $ N $\rightarrow $ CN and A $\rightarrow $ AP to create the common noun “board game” and the adjective phrase first. The next rule is AP $\rightarrow $ CN $\rightarrow $ CN. The last rule to be applied is CN $\rightarrow $ NP. The selection is easily decided since the input and the output of the rules are pre-determined, and there is no ambiguity in the selection process.

The encoder uses the GF rules and the components identified by the previous subsections to produce different constructors for different components of a sentence. A part of the output of the GF encoder for the object “game” is

The encoder will also create the operators that will be included in the oper section of the GF grammar for supporting the new constructor. For example, the following operators will be generated for serving the Game constructor above:

Method ::: GF Grammar Exporter

The GF Grammar Exporter has the simplest job among all modules in the system. It creates a GF program for a paragraph using the GF grammars created for the sentences of the paragraph. By taking the union of all respective elements of each grammar for each sentence, i.e., categories, functions, linearizations and operators, the Grammar Exporter will group them into the set of categories (respectively, categories, functions, linearizations, operators) of the final grammar.

Experiments

We describe our method of generating natural language in two applications. The first application is to generate a natural language description for workflow created by the system built in the Phylotastic project described in BIBREF2. Instead of requiring that the ontologies are annotated using Attempto, we use natural language sentences to annotate the ontologies. To test the feasibility of the approach, we also conduct another use case with the second ontology, that is entirely different from the ontologies used in the Phylotastic project. The ontology is about people and includes descriptions for certain class.

The second application targets the challenge of creating an abstract Wikipedia from the BlueSky session of 2018 International Semantic Web Conference BIBREF7. We create an intermediate representation that can be used to translate the original article in English to another language. In this use case, we translate the intermediate representation back to English and measure how the translated version stacks up again the original one. We assess the generation quality automatically with BLEU-3 and ROUGE-L (F measure). BLEU BIBREF16 and ROUGE BIBREF17 algorithms are chosen to evaluate our generator since the central idea of both metrixes is “the closer a machine translation is to a professional human translation, the better it is”, thus, they are well-aligned with our use cases' purpose. In short, the higher BLUE and ROUGE score are, the more similar the hypothesis text and the reference text is. In our use case, the hypothesis for BLEU and ROUGE is the generated English content from the intermediate representation, and the reference text is the original text from Wikipedia.

Experiments ::: NLG for Annotated Ontologies

As described in BIBREF2, the author's system retrieves a set of atoms from an ASP program such as those in Listing where phylotastic FindScientificNamesFromWeb GET was shortened to service, propagates the atoms, and constructs a set of sentences having similar structure to the sentence “The input of phylotastic FindScientificNamesFromWeb GET is a web link. Its outputs are a set of species names and a set of scientific names”. In this sentence, phylotastic FindScientificNamesFromWeb GET is the name of the service involved in the workflow of the Phylotastic project. All of the arguments of the atoms above are the names of classes and instances from Phylotastic ontology.

We replace the original Attempto annotations with the natural language annotations as in Table TABREF24 and test with our system.

With the same set of atoms as in Listing , our system generates the following description “Input of phylotastic FindScientificNamesFromWeb GET is web link. Type of web link is url. Output of phylotastic FindScientificNamesFromWeb GET is scientific names. Output of phylotastic

FindScientificNamesFromWeb GET is species names. Type of scientific names is names. Type of species name is names.”.

We also test our system with the people ontology as noted above. We extract all comments about people and replace compound sentences with simple sentences, e.g., “Mick is male and drives a white van” is replaced by the two sentences “Mick is male” and “Mick drives a white van.” to create a collection of sample sentences. We then use our system to generate a GF program which is used to generate sentences for RDF tuples. Sample outputs for some tuples are in Table TABREF25. This shows that for targeted applications, our system could do a reasonable job.

Experiments ::: Intermediate Representation for Wiki Pages

Since our system creates a GF program for a set of sentences, it could be used as an intermediate representation of a paragraph. This intermediate representation could be used by GF for automatic translation as GF is well-suited for cross-languages translation. On the other hand, we need to assess whether the intermediate representation is meaningful. This use case aims at checking the adequacy of the representation. To do so, we generate the English sentences from the GF program and evaluate the quality of these sentences against the original ones. We randomly select 5 articles from 3 Wikipedia portals: People, Mathematics and Food & Drink.

With the small set of rules introducing in this paper to recognize sentence structure, there would be very limited 4-gram in the generated text appearing in original Wikipedia corpus. Therefore, we use BLEU-3 with equal weight distribution instead of BLEU-4 to assess the generated content. Table TABREF27 shows the summary of the number of assessable sentences from our system. Out of 62 sentences from 3 portals, the system cannot determine the structure 2 sentences in Mathematics due to their complexity. This low number of failure shows that our 5 proposed sentence structures effectively act as a lower bound on sentence recognition module.

In terms of quality, Table TABREF28 shows the average of BLEU and ROUGE score for each portal. Note that the average BLUE score is calculated only on BLEU assessable sentences, while average ROUGE score is calculated on the sentences whose structure can be recognized and encoded by our system. We note that the BLEU or ROUGE score might not be sufficiently high for a good quality translation. We believe that two reasons contribute to this low score. First, the present system uses fairly simple sentence structures. Second, it does not consider the use of relative clauses to enrich the sentences. This feature will be added to the next version of the system.

Table TABREF32 summarizes the result of this use case. On the left are the paragraphs extracted from the Wikipedia page about Rice in Food & Drink, Decimal in Mathematics, and about Alieu Ebrima Cham Joof from People. As we can see, the main points of the paragraphs are maintained.

Related Works

The systems developed in BIBREF18, BIBREF19, BIBREF3 use statistical generation method to produce descriptions of tables or explanation and recommendation from users' reviews of an item. All three systems are capable of generating high quality descriptions and/or explanations. In comparing to these systems, our system does not use the statistical generation method. Instead, we use Grammatical Framework for the generation task. A key difference between these systems and our system lies in the requirement of a large corpus of text in a specific domain for training and generation of these systems. Our system can work with very limited data and a wide range of domains.

Another method for generating natural language explanation for an question-answering system is proposed in BIBREF20, BIBREF4. BIBREF20 (BIBREF20) describes a system that can give reasonable and supportive evidence to the answer to a question asked to an image, while BIBREF4 (BIBREF4) generates explanations for scheduling problem using argumentation. BIBREF21 (BIBREF21) use ASP to develop a system answering questions in the do-it-yourself domain. These papers use templates to generate answers. The generated GF program generated by our system, that is used for the NLG task, is automatically created from a provided input.

The sophisticated system presented by BIBREF5 translates both question and the given natural language text to logical representation, and uses logical reasoning to produce the answer. Our system is similar to their system in that both employ recent developments of NLP into solving NLG problems.

Conclusions and Future Work

We propose a system implemented using answer set programming (ASP) and Grammatical Framework (GF), for automatic generation of natural language descriptions in applications targeting mainstream users. The system does not require a large corpus for the generation task and can be used in different types of applications.

In the first type of applications, the system can work with annotated ontologies to translate a set of atoms—representing the answer to a query to the ontology—to a set of sentences. To do so, the system extracts the annotations related to the atoms in the answer and creates a GF program that is then used to generate natural language description of the given set of atoms. In the second type of applications, the system receives a paragraph of text and generates an intermediate representation—as a GF program—for the paragraph, which can be used for different purpose such as cross-translation, addressing a need identified in BIBREF7 .

Our use cases with different ontologies and Wikipedia portals provide encouraging results. They also point to possible improvements that we plan to introduce to the next version of the system. We will focus on processing relative clauses and enriching the set of sentence structures, especially for compound and complex sentences.