International Journal of Computational Linguistics and Applications
IJCLA VOL. 5, NO. 1, JAN-JUN 2014, PP. 135–158
RECEIVED 13/04/14 ACCEPTED 02/06/14 FINAL 24/06/14

Should Syntactic N-grams Contain Names of Syntactic Relations?

GRIGORI SIDOROV

Instituto Politécnico Nacional, Mexico

ABSTRACT

In this paper, we discuss a specific type of mixed syntactic n-grams: syntactic n-grams with relation names, snr-grams. This type of syntactic n-grams combines lexical elements of the sentence with the syntactic data, but it keeps the properties of traditional n-grams and syntactic n-grams. We discuss two possibilities related to labelling of the relation names for snr-grams: based on dependencies and based on constituencies. Examples of various types of n-grams, sn-grams, and snr-grams are given.

1 INTRODUCTION

In our previous works starting in 2012 we proposed a concept of syntactic n-grams (Sidorov, Velasquez, Stamatatos, Gelbukh & Chanona-Hernandez 2012, 2013, 2014; Sidorov 2013a, 2013b, 2013c). This concept is quite on the agenda of the computational linguistics: say, our works obtained many positive feedback comments, besides, the same concept was implemented independently for English language in the form of a large collection of syntactic n-grams obtained from books by (Goldberg and Orwant, 2013), while they were working on this project in Google.

Let us remind that syntactic n-grams are n-grams of textual elements obtained in a specific non-linear manner based on syntactic relations (Sidorov 2013c), i.e., instead of using the order of elements in the surface structure, the syntactic structure is used. For obtaining syntactic n-grams, we traverse the syntactic tree and use the order of elements in it. It is equivalent (but probably less clear) to say that we use subtrees of a syntactic tree as syntactic n-grams. It is obvious that the syntactic structure is non-linear with respect to the surface structure: the order of elements is usually changed. We discuss the concept of syntactic n-grams in greater detail in the next section.

Note that syntactic n-grams can be used in any task in the field of the Natural Language Processing, when traditional n-grams can be applied. It is especially important in the modern paradigm related to application of machine learning algorithms, because this paradigm is completely based on the concept of vector space model and feature selection, where the features are precisely n-grams or syntactic n-grams.

Machine learning simulates human ability for classification of objects based on their similarity. What the best features selected for similarity calculus are, depends on every specific task, for example, for thematic classification of documents we need to take into account words that are thematically related to each topic and ignore auxiliary words, while, say, for analysis of author's writing style we would prefer to focus precisely on auxiliary words, because they may reflect the style. Both supervised and unsupervised machine learning algorithms can be applied using syntactic n-grams as features in the corresponding vector space model.

An alternative to machine learning methods is the paradigm based on formulation and application of hand crafted rules. This paradigm was prevalent until the end of the 20th century (Bolshakov, Gelbukh 2004). In this paradigm, the human evaluators analyze the example data of the problem, try to propose some hypotheses about the structure and function of the phenomena related to the problem and after this extract problem-dependent features and formulate rules. These rules usually correspond to selectional preferences, i.e., the generalized restrictions on combination between elements. The current state of the art is that machine learning algorithms—if they have sufficiently large marked corpus for training—outperform human crafted rules. Note that the human effort is still present, though it is moved from formulation of the rules to marking of the corpora (Gelbukh 2013).

The advantage of machine learning algorithms over humans is that these algorithms are consistent and consider many variants during feature selection using vast data, while humans are not consistent, cannot process big volumes of data, and cannot generalize over too many examples. Obviously, the humans are better than the computers while marking the corpora using intuition, because they can use the extra linguistic world knowledge and common sense, which computers do not possess, for understanding of individual sentences or texts. But it seems that given a marked corpus, a machine learning algorithm can perform better feature selection than a human.

The rest of the paper is organized as follows. Dependency and constituency representations of syntactic relations are discussed in Section 2. In Section 3, we describe the concept of syntactic n-grams and present their various types. In Section 4 we propose the concept of syntactic n-grams with relation names (snr-grams) and give some examples of their extraction using formalisms of dependencies and constituencies. Finally, conclusions are drawn in Section 5.

2 CONSTITUENCIES VS. DEPENDENCIES AS SYNTACTIC REPRESENTATIONS

There are two main formalisms for representation of syntactic structure: dependencies and constituents. The dependency formalism directly reflects relations between words, usually using arrows. Since one word in a syntactic relation is the head word, while the other one is the dependent word, the arrow has the direction: head→dependent. The arrows are labelled with the types of syntactic relations. If there is no natural head, like, say, in case of a coordinative relation, some decision about the head/dependent words should be made anyway.

The constituency formalism represents syntactic relations with respect to the underlying formal grammar and reflects the history of the syntactic tree derivation according to this grammar. The syntactic relations between words are established on the basis of the applied grammar rules: derivation history. Note that some relations are established not between words themselves, but between constituents, which represent the result of the previous application of the rules.

Constituency trees have longer history in usage in the computational linguistics, because they are directly related to application of generative grammars (N. Chomsky). Modern approaches pay more attention to dependency trees, because they are more natural and direct. Besides, they contain the information about the syntactic roles of words, like “direct object”, “subject”, etc.

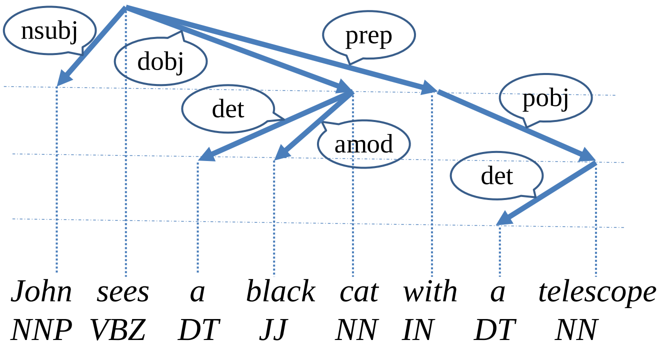
nsubj
dobj
prep
det
amod
pobj
det
John sees a black cat with a telescope
NNP VBZ DT JJ NN IN DT NN

Fig. 1. Example of a dependency tree.

2.1 Example of the Representation of a Syntactic Tree

Let us present an example of the dependency and constituency formalisms for a syntactic tree, for instance, for the phrase *John sees a black cat with a telescope*. The syntactic tree that uses dependency formalism is shown in Fig. 1. We also show the POS tags of each word on the next line below the corresponding word.

The example of representation of the same phrase using the formalism of constituencies is shown in Fig. 2. In this case, we mark with wider line the part of the constituent that corresponds to the head word. We also show in the tree structure the left parts of the applied rules, i.e., the generalization introduced by each rule.

This constituency tree is generated by the following very simple formal grammar. It is clear that real parsers can use more complex or more general rules, but for our discussion this grammar is sufficient. We mark with “*” the head elements in the rules.

S → NNP VP*
VP → VP* PP
NP → JJ NN*
NP → DT NN*
NP → DT NP*
PP → IN* NP

VP → VBZ* NP

The derivation history of the phrase is the order of application of the grammar rules. For example, we start with the rules that correspond directly to words (terminal nodes) “NP → DT NN*”, “NP → JJ NN*”. After this, the “intermediate” rules like “VP → VBZ* NP” are applied and finally the “top” rule “S → NNP VP*” is used. This derivation history corresponds to the analysis strategy “bottom-up”, being the other possible strategy the reverse order of application of the rules: “top-down”.

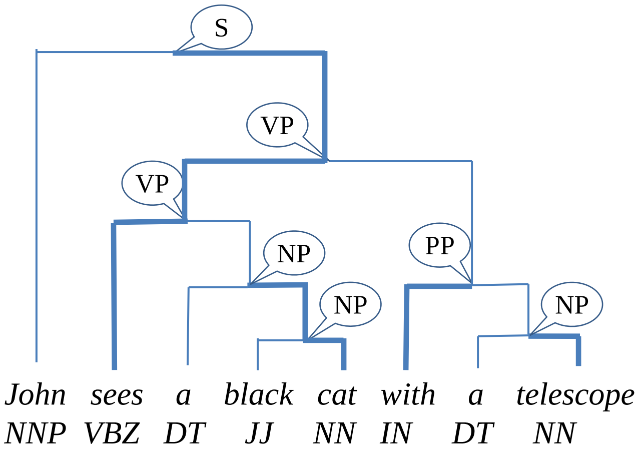
S
VP
VP
NP
PP
NP
NP
John sees a black cat with a telescope
NNP VBZ DT JJ NN IN DT NN

Fig. 2. Example of a constituency tree.

2.2 Conversion between Constituencies and Dependencies

It is well-known that dependency and constituency formalisms are equivalent in general, i.e., there exists an algorithm that transforms the dependency tree structure into the constituency tree structure and vice versa (Gelbukh, Calvo, Torres 2005). It is not surprising, because both types of trees reflect the same syntactic reality. Note that this is only general (structural) conversion, as it does not convert the syntactic labels in both directions.

The algorithm for constituency to dependency general conversion is simple. For each word that is a head word (it is marked with “wider” line) go up in the tree. At each step (while going up following the constituents) go down to a dependent constituent. After this follow downwards the head relations only (the “wider” line) and draw the arrow from the head word to the obtained dependent word. Continue going up in the tree from the point, when you start going down.

For constituency to dependency general conversion, the formal grammar should mark the words that are heads on the right side of the rules, because otherwise we would not know the directions of the dependency arrows. Note that if the grammar does not mark them, the marking can be done in a random manner, but obviously with not so good results: the conversion will be done, but some arrows would have anti-intuitive directions. It is also clear that the resulting dependency tree does not contain the names of syntactic relations for the arrows.

The algorithm for dependency to constituency general conversion is also simple. We start with arrows at the lowest level and go to upper levels. For each arrow we establish a constituent relation for the pair of words, being the head word the starting point of the arrow. If the head word already forms a constituent, then this constituent should be used instead of the word itself. Some additional conventions are necessary, for example, in case of bifurcations, we first process the arrows that are the closest ones to the word, or that *nsubj* relation is processed last.

It is clear that for dependency to constituency general conversion the resulting constituency tree does not have the interpretation of constituents (left parts of the rules represented in the tree structure), because it is precisely what the formal grammar does; in certain sense, the resulting representation will lack of generalization for constituents.

3 SYNTACTIC N-GRAMS

As we mentioned above, we introduced the concept of syntactic n-grams in our previous works (Sidorov et al. 2012, 2013, 2014; Sidorov 2013a, 2013b, 2013c). Similar ideas were proposed in (Pado, Lapata 2007; Gelbukh 1999), but they were treated as very specific methods for certain tasks of syntactic or semantic analysis. The importance of the concept is confirmed by the fact that Google obtained and made public syntactic n-grams for a large set of books in English (Goldberg and Orwant, 2013).

In our early works we preferred to use the term "syntactic dependency based n-grams", adding the words "dependency based". It was important, because there is possible naive misinterpretation of the term "syntactic n-grams" as "sequence of POS tags", because POS tags are perceived as carrying some syntactic information. In fact, it is not true: POS tags are more morphological than syntactic phenomena—the syntactic information is used only for disambiguation between several possible POS tags for a word. At most, we can consider them as morphosyntactic entities. Now, as the term "syntactic n-grams, sn-grams" is more habitual, we can omit the words "dependency based". Note that we say "dependency based" (and not "constituency based"), because syntactic dependencies are much more direct projection of syntactic paths for construction of sn-grams. Constituencies can be applied to construction of sn-grams as well, though not so naturally, see discussion in Sections 2 and 4.

So, while traditional n-grams are sequences of textual elements (words, POS tags, etc.) taken as they appear in texts, the general idea behind syntactic n-grams is to take the surface textual elements in a non-linear order by following paths in syntactic trees. In this case, the order of textual elements is usually changed in comparison with the surface structure.

3.1 Types of Syntactic N-grams

In our previous works, we have proposed the classification of syntactic n-gram types. Depending on the elements that constitute them, there can be syntactic n-grams of words/lemmas/stems (lexical elements), POS tags, SR tags (names of Syntactic Relations), multiword expressions (Gelbukh, Kolesnikova 2013a, 2013b; Ledeneva, Gelbukh, García-Hernández 2008), and even of characters (Sidorov et al. 2013, 2014).

For obtaining character sn-grams, we first construct sn-gram of lexical units (words or lemmas) and then character sn-grams are constructed over this sequence in the same way as it is done for traditional character n-grams. Note that for this procedure it is preferable to use sn-grams that contain most number of elements (long sn-grams, n-grams with large values of n), but each lexical element should be considered at least once. There is a problem for future research: how to calculate correctly the frequencies of character sn-grams obtained in this manner, because many elements in sn-grams are repeated.

There also can be mixed sn-grams, for example, one element in an sn-gram is a POS tag and the other one is a lexical unit. Note that character sn-grams cannot be naturally mixed with other types of sn-grams, because they have different nature: all other types of sn-grams reflect properties of words (lexical unit, POS tag), even SR tags reflect the relations of a word with other word, while character sn-grams are sequences of characters obtained from already existing sn-grams of lexical units, so they are derivate, and in a certain sense they are secondary. We insist on considering them because in certain tasks, like, for example, authorship attribution, traditional character n-grams quite surprisingly give very good results, so character sn-grams should be tried as well.

On the other hand, in (Sidorov, 2013a) we have proposed differentiating between continuous (non-interrupted path, path without bifurcations) and non-continuous (path with interruptions or returns (or bifurcations)) syntactic n-grams. It is obvious that continuous sn-grams are a special type of non-continuous sn-grams, namely sn-grams with no returns (without bifurcations). Intuitively, we consider that continuous sn-grams can contain more important linguistic information, but it should be verified for various tasks in the experimental manner. It is clear that for syntactic bigrams there is no difference between continuous and non-continuous sn-grams, because no bifurcations are possible in case of exactly two elements in an n-gram.

Note that here appears another possible naive misinterpretation of the general term “syntactic n-grams” that would be “n-grams of names of syntactic relations (SR tags)”. It is possible, because these sn-grams can be obtained only if we apply parsing before. Nevertheless, this interpretation is too narrow: yes, it is the possible type of sn-grams, but there are other types as well. In general, while speaking about syntactic n-grams we refer not to a specific type of elements (SR tags, words, etc.), but to the manner of their construction by following paths in syntactic trees.

Another important consideration is related to the traditional practice of treatment of stop words (auxiliary words). There are two possibilities: taking them into account vs. filtering out of stop words. The possibility of filtering out of stop words can be easily applied to syntactic n-grams: we should follow syntactic paths and when we encounter with a stop word, we ignore it and just continue with the next word according to the path. In fact, this idea was generalized as "filtered n-grams" in (Sidorov 2013c): we can filter out not only stop words, but any words that do not comply with any chosen criterion, for example, it can be a thresholds based on tf-idf values.

Finally, we would like to mention that the elements of the same level in an sn-gram can be taken as they appear in the sentence, or can be reordered according to some criteria, for example, using the alphabetic order of the elements. The first possibility takes into account the word order in the sentences, while the second one tries to ignore possible (insignificant) changes in the word order.

3.2 Extraction of Syntactic N-grams and their Representation

The software for extraction of syntactic n-grams is available on the Web page of the author[1]. It takes as the input the file generated by the Stanford parser (de Marneffe, MacCartney, Manning 2006) and it produces sn-grams of the desired size and type.

Note that the software also treats in a practical manner the problem of exponential growth of the number of sn-grams in case of too many dependents of a word. This problem consists in the fact that if the number of the dependent words is large, say, more than six, then the number of possible combinations (i.e., non-continuous sn-grams) may become too large. It is very rare situation to have so many dependent words, but it may appear in real life, especially if something went wrong with parsing or if we want to treat punctuation (like parenthesis) and the parser chooses one word as their head.

For the example in Fig. 1 and Fig. 2, the Stanford parser generates the output presented in Fig. 3 and Fig. 4 correspondingly.

Let us now discuss how to represent syntactic n-grams. If we use only continuous sn-grams, then we can represent them using the sequences of words just like in case of the traditional n-grams. But if we start considering non-continuous sn-grams, then it turns out that we need special metalanguage for their representation, namely for distinguishing the words that form a sequence from the words that have returns in the path (bifurcations).

For example, if we have three words A, B, C and we want to express that both B and C are dependent from A, i.e., there is a return in the path (a bifurcation), then we separate B and C with a comma and

[1] http://www.cic.ipn.mx/~sidorov

```
nsubj(sees-2, John-1)
root(ROOT-0, sees-2)
det(cat-5, a-3)
amod(cat-5, black-4)
dobj(sees-2, cat-5)
prep(sees-2, with-6)
det(telescope-8, a-7)
pobj(with-6, telescope-8)
```

Fig. 3. Results of the analysis using dependencies.

```
(ROOT
 (S
  (NP (NNP John))
  (VP (VBZ sees)
   (NP (DT a) (JJ black) (NN cat))
   (PP (IN with)
    (NP (DT a) (NN telescope))))
  ))
```

Fig. 4. Results of the analysis using constituencies.

put them into the brackets: “A [B, C]”. If there is no bifurcation that means that C depends from B and B depends from A, then we just write “A B C”. In the current version of our software we add more brackets: “[A[B[C]]]” and “[A[B,C]]”. This notation reflects more consistent use of brackets in each node and better shows the underlying tree structure. Note that if used uniformly, it does not affect the identity of sn-grams.

3.3 Example of Extraction of Syntactic N-grams

Let us consider the example presented in Fig. 1. The second line of the example contains the POS tags for each word. It is obvious that we can substitute lexical units with their lemmas, for example, use *see* instead of *sees,* as well as with their POS tags, for instance, use VBZ instead of *sees* or NN instead of *telescope*, etc. Thus, we suppose that the reader understands this possibility and we will not illustrate it in the figure and in further discussion: we should just remember that while we use words, they can as well be substituted by lemmas or POS tags, or any combinations of these elements. As we mentioned in our previous works, it is a question of future experimental research to determine what types of sn-grams or mixed sn-grams are useful for particular tasks.

Let us extract all possible traditional n-grams and syntactic n-grams of various sizes and types from the example sentence. First, we present traditional n-grams of words of various sizes and syntactic n-grams of the same sizes in Tables 1-6. We start with bigrams and go till 7-grams. Note that in practical tasks of the computational linguistics, we usually do not need larger size of n-grams, because they do not repeat any more in texts, i.e., their frequency is always equal to 1 in any corpus and they are practically useless.

Table 1. Traditional and syntactic bigrams.

Traditional bigrams	Syntactic bigrams
John sees	*sees[with]*
sees a	*telescope[a]*
a black	*sees[cat]*
black cat	*cat[black]*
cat with	*with[telescope]*
with a	*cat[a]*
a telescope	*sees[John]*

Table 2. Traditional and syntactic trigrams.

Traditional trigrams	Syntactic trigrams
John sees a	*with[telescope[a]]*
sees a black	*sees[cat,with]*
a black cat	*sees[John,cat]*
black cat with	*sees[John,with]*
cat with a	*sees[cat[a]]*
with a telescope	*sees[with[telescope]]*
	cat[a,black]
	sees[cat[black]]

Table 3. Traditional and syntactic 4-grams.

Traditional 4-grams	Syntactic 4-grams
John sees a black	*sees[cat[a,black]]*
sees a black cat	*sees[John,with[telescope]]*
a black cat with	*sees[cat[a],with]*
black cat with a	*sees[John,cat[black]]*
cat with a telescope	*sees[John,cat,with]*
	sees[cat,with[telescope]]
	sees[John,cat[a]]
	sees[cat[black],with]
	sees[with[telescope[a]]]

Table 4. Traditional and syntactic 5-grams.

Traditional 5-grams	Syntactic 5-grams
John sees a black cat	*sees[John,cat[black],with]*
sees a black cat with	*sees[cat[a,black],with]*
a black cat with a	*sees[John,with[telescope[a]]]*
black cat with a telescope	*sees[John,cat[a,black]]*
	sees[cat[black],with[telescope]]
	sees[cat[a],with[telescope]]
	sees[John,cat,with[telescope]]
	sees[cat,with[telescope[a]]]
	sees[John,cat[a],with]

Table 5. Traditional and syntactic 6-grams.

Traditional 6-grams	Syntactic 6-grams
John sees a black cat with	*sees[John,cat[a,black],with]*
sees a black cat with a	*sees[John,cat[a],with[telescope]]*
a black cat with a telescope	*sees[John,cat,with[telescope[a]]]*

	sees[cat[black],with[telescope[a]]]
	sees[cat[a],with[telescope[a]]]
	sees[John,cat[black],with[telescope]]
	sees[cat[a,black],with[telescope]]

Table 6. Traditional and syntactic 7-grams.

Traditional 7-grams	Syntactic 7-grams
John sees a black cat with a	*sees[John,cat[a],with[telescope[a]]]*
sees a black cat with a telescope	*sees[cat[a,black],with[telescope[a]]]*
	sees[John,cat[black],with[telescope[a]]]
	sees[John,cat[a,black],with[telescope]]

It can be observed that syntactic n-grams are much more linguistically motivated, because for their construction we use very important linguistic knowledge: syntactic structure. For example, traditional n-grams like “with a” or “sees a” no longer form part of the features for machine learning algorithms. A counterargument might be that these n-grams can appear consistently in the corpus. The answer to this counterargument is that though it is true, these n-grams contain more noise than real information, because there is no linguistic reality behind them.

Now let us present syntactic n-grams of SR tags, Tables 7-12. Obviously, there are no traditional n-grams that use this type of elements.

Table 7. Syntactic bigrams of SR tags.

Syntactic bigrams
prep[pobj]
root[nsubj]
root[prep]
root[dobj]
dobj[amod]

pobj[det]
dobj[det]

Table 8. Syntactic trigrams of SR tags.

Syntactic trigrams
prep[pobj[det]]
root[dobj,prep]
root[dobj[amod]]
root[nsubj,dobj]
dobj[det,amod]
root[prep[pobj]]
root[nsubj,prep]
root[dobj[det]]

Table 9. Syntactic 4-grams of SR tags.

Syntactic 4-grams
root[dobj,prep[pobj]]
root[nsubj,dobj,prep]
root[prep[pobj[det]]]
root[dobj[det],prep]
root[nsubj,dobj[amod]]
root[dobj[amod],prep]
root[nsubj,prep[pobj]]
root[nsubj,dobj[det]]
root[dobj[det,amod]]

Table 10. Syntactic 5-grams of SR tags.

Syntactic 5-grams
root[dobj[amod],prep[pobj]]

root[dobj,prep[pobj[det]]]
root[nsubj,dobj[amod],prep]
root[dobj[det,amod],prep]
root[nsubj,dobj[det,amod]]
root[dobj[det],prep[pobj]]
root[nsubj,prep[pobj[det]]]
root[nsubj,dobj,prep[pobj]]
root[nsubj,dobj[det],prep]

Table 11. Syntactic 6-grams of SR tags.

Syntactic 6-grams
root[nsubj,dobj[amod],prep[pobj]]
root[dobj[amod],prep[pobj[det]]]
root[dobj[det,amod],prep[pobj]]
root[nsubj,dobj[det,amod],prep]
root[dobj[det],prep[pobj[det]]]
root[nsubj,dobj[det],prep[pobj]]
root[nsubj,dobj,prep[pobj[det]]]

Table 12. Syntactic 7-grams of SR tags.

Syntactic 7-grams
root[nsubj,dobj[det,amod],prep[pobj]]
root[nsubj,dobj[amod],prep[pobj[det]]]
root[dobj[det,amod],prep[pobj[det]]]
root[nsubj,dobj[det],prep[pobj[det]]]

In a similar manner, we can construct syntactic n-grams using constituency trees, namely, the derivation history, on the basis of considerations presented in section 4. We give the example of bigrams of relations based on derivation history in Table 13. The parentheses are used for containing the corresponding fragment of the derivation history. In this case we consider that the relation "root" corresponds to the left part of the rule with the element "S". For comparison, we give also the syntactic bigrams based on SR tags from Table 7.

Table 13. Syntactic bigrams of derivation history fragments.

Syntactic bigrams based on derivation history	Syntactic bigrams based on SR tags
(VP,VP,PP)[(PP,NP)]	*prep[pobj]*
(S)[(NN)]	*root[nsubj]*
(S)[(VP,VP,PP)]	*root[prep]*
(S)[(VP,NP,NP)]	*root[dobj]*
(VP,NP,NP)[(NP)]	*dobj[amod]*
(PP,NP)[(NP)]	*pobj[det]*
(VP,NP,NP)[(NP)]	*dobj[det]*

4 Syntactic N-grams with Relation Names (Snr-grams)

We hope that the reader now has clear idea about the concept of syntactic n-grams and their types. Among types of syntactic n-grams we mentioned that there can be mixed syntactic n-grams. In this sense, we already considered the fact that syntactic n-grams can contain names of syntactic relations mixed with other elements. Nevertheless, there are considerations for drawing attention to this particular type of sn-grams: they contain both lexical/morphological elements (words, lemmas, POS tags) and at the same time the names of syntactic relations (SR tags). Let us call these sn-grams that contain relation names "snr-grams", when we prefer to use the abbreviation.

It can be observed that snr-grams convey more information than any other type of n-grams or sn-grams, and still they can be used as features in machine learning tasks, when other n-grams can be used. So, we believe that this type of sn-grams deserves special attention. We have preliminary information that snr-grams performed better in the task of the periphrasis as compared to n-grams and other types of sn-grams (personal communication of Hiram Calvo, the corresponding paper will be published soon).

There is also a certain problem that consists in how to count the number of elements in snr-grams. If we count both words/POS tags together with SR tags then, say, there will be no bigrams, and in general, no n-grams with even values of n. So our suggestion, if we deal with snr-grams, is counting only the elements different from SR tags. In case that we deal with sn-grams of SR tags only, then, obviously, we should count these elements (SR tags). In general, if we want to use mixed n-grams, when certain elements are word based (words, POS tags) and the other elements are relation based (SR tags), we should count SR tags only if we do not want to take into account the word based elements. For example, if we want to consider syntactic bigrams, where the first element is the word and the second one is the SR tag, then we treat the SR tags as the proper element of the bigrams. On the other hand, if we are working with snr-grams, then SR tags should not be counted for determining the snr-gram size.

Let us present snr-grams from the example above using SR tags as part of snr-grams, Tables 14-16. We use parentheses to contain the relation name, which is placed before each word. Note that it should appear immediately before the word because of ambiguities of possible bifurcations.

There are two possibilities for the first word in an snr-gram:

1. We can add to the first word of an snr-gram the corresponding SR tag (the name of the corresponding incoming arrow), because it always exists, or
2. We can leave the first word in an snr-gram without the SR tag, because it does not connect this word to any other element of the given snr-gram.

We choose the second option. For example, instead of the bigram *(pobj)telescope[(det)a]*, we write the bigram *telescope[(det)a]*.

Table 14. Snr-grams of size 2 (SR tags).

Snr-grams
sees[(prep)with]
telescope[(det)a]

sees[(dobj)cat]
cat[(amod)black]
with[(pobj)telescope]
cat[(det)a]
sees [(nsubj)John]

Table 15. Snr-grams of size 3 (SR tags).

Snr-grams
with[(pobj)telescope[(det)a]]
sees[(dobj)cat,(prep)with]
sees[(nsubj)John,(dobj)cat]
sees[(nsubj)John,(prep)with]
sees[(dobj)cat[(det)a]]
sees[(prep)with[(pobj)telescope]]
cat[(det)a,(amod)black]
sees[(dobj)cat[(mod)black]]

Table 16. Snr-grams of size 4 (SR tags).

Snr-grams
sees[(dobj)cat[(det)a,(amod)black]]
sees[(nsubj)John,(prep)with[(pobj)telescope]]
sees[(dobj)cat[(det)a],(prep)with]
sees[(nsubj)John,(dobj)cat[(amod)black]]
sees[(nsubj)John,(dobj)cat,(prep)with]
sees[(dobj)cat,(prep)with[(pobj)telescope]]
sees[(nsubj)John,(dobj)cat[(det)a]]
sees[(dobj)cat[(amod)black],(prep)with]
sees[(prep)with[(pobj)telescope[(det)a]]]

In the tables above, we used dependency trees for extraction of snr-grams, but constituency trees can be used as well. There are several possibilities related to which part of the derivation history of the corresponding constituency tree should be included into the description of each relation:

- Use the derivation history that is below the node vs. above the node vs. both parts (above and below). These strategies correspond to bottom-up parsing and top-down parsing.
- Use only the left part of the rule vs. use the whole rule,
- Use only the last derivation vs. use the whole derivation chain or several last steps (say, two, three, etc.).

We present the example for (1) the whole derivation chain, (2) below the node, and (3) using the left part of the rule. Other possibilities should be tried as well in experiments for particular tasks. We start from the left element of a constituent, go up to the least common node, and then go down to the right element. At each step we take the left part of the corresponding rule. In tables 17-19 we present the snr-grams of sizes 2, 3, and 4 extracted from the example sentence.

Table 17. Snr-grams of size 2 (derivation history).

Snr-grams based on constituencies
sees[(VP,VP,PP)with]
telescope[(NP)a]
sees[(VP,NP,NP)cat]
cat[(NP)black]
with[(PP,NP)telescope]
cat[(NP)a]
sees [(S,VP,VP)John]

Table 18. Snr-grams of size 3 (derivation history).

Snr-grams based on constituencies
with[(PP,NP)telescope[(NP)a]]

sees[(VP,NP,NP)cat,(VP,VP,PP)with]
sees[(S,VP,VP)John,(VP,NP,NP)cat]
sees[(S,VP,VP)John,(VP,VP,PP)with]
sees[(VP,NP,NP)cat[(NP)a]]
sees[(VP,VP,PP)with[(PP,NP)telescope]]
cat[(NP)a,(NP)black]
sees[(VP,NP,NP)cat[(NP)black]]

Table 19. Snr-grams of size 4 (derivation history).

Snr-grams based on constituencies
sees[(VP,NP,NP)cat[(NP)a,(NP)black]]
sees[(S,VP,VP)John,(VP,VP,PP)with[(PP,NP)telescope]]
sees[(VP,NP,NP)cat[(NP)a],(VP,VP,PP)with]
sees[(S,VP,VP)John,(VP,NP,NP)cat[(NP)black]]
sees[(S,VP,VP)John,(VP,NP,NP)cat,(VP,VP,PP)with]
sees[(VP,NP,NP)cat,(VP,VP,PP)with[(PP,NP)telescope]]
sees[(S,VP,VP)John,(VP,NP,NP)cat[(NP)a]]
sees[(VP,NP,NP)cat[(NP)black],(VP,VP,PP)with]
sees[(VP,VP,PP)with[(PP,NP)telescope[(NP)a]]]

If we use SR tags, then the usefulness of snr-grams is explained by the fact that they allow to distinguish the syntactic role of each element in the n-gram, for example, "*sees[(nsubj)John]*" vs. "*sees[(dobj)cat]*", when the only difference in the verb-noun combination is the relation name. Obviously, it depends on the task if this difference is relevant or not.

In case of derivation history fragments, their function is not so clear as in case of SR tags, for example *sees[(S,VP,VP)John]* vs. *sees[(VP,NP,NP)cat]*. We can deduce that one of the fragments includes the tree root ("S"), while the other does not. We can also see how far is the distance between these two nodes in terms of the number of the applied rules and their types. Future experiments should demonstrate how useful this information is.

5 Conclusions

In this paper we introduced and discussed the concept of the syntactic n-grams with relation names, snr-grams, which is a special type of mixed syntactic n-grams. We presented examples of snr-grams of various sizes, constructed for both tags of names of syntactic relations (SR tags) and for fragments of derivation history. We consider that snr-grams can be applied in many tasks of the Natural Language Processing as features for machine learning algorithms. Future experiments should confirm in which tasks their usage is beneficial.

For having the possibility of discussion of the concept of the snr-grams, we described the formalisms of dependencies and constituents used for the representation of the syntactic information and several related algorithms. We also described several issues related to the introduced in our previous works concept of syntactic n-grams.

Acknowledgements This work was done under partial support of the Mexican Government (CONACYT, SNI, COFAA-IPN, SIP-IPN 20144274) and FP7-PEOPLE-2010-IRSES: “Web Information Quality – Evaluation Initiative (WIQ-EI)” European Commission project 269180.

References

1. Bolshakov, I.A., Gelbukh, A.: Computational linguistics: Models, resources, applications. IPN–UNAM–FCE, (2004) 187 pp.
2. Calvo, H., Gelbukh, A. Improving Prepositional Phrase Attachment Disambiguation Using the Web as Corpus. Lecture Notes in Computer Science 2905 (2003) 604–610
3. de Marneffe, M.C., MacCartney, B. Manning, C.D.: Generating typed dependency parses from phrase structure parses. In: Proceedings of LREC 2006 (2006)
4. Gelbukh, A.: Natural language processing: Perspective of CIC-IPN. In: Proceedings of the International Conference on Advances in Computing, Communications and Informatics, ICACCI 2013, IEEE (2013) 2112–2121
5. Gelbukh, A.: Syntactic disambiguation with weighted extended subcategorization frames. In: Proceedings of PACLING-99, Pacific Association for Computational Linguistics, University of Waterloo, Canada (1999) 244–249

6. Gelbukh, A., Calvo, H. Torres, S.: Transforming a Constituency Treebank into a Dependency Treebank. Procesamiento de Lenguaje Natural 35 (2005) 145-152
7. Gelbukh, A., Kolesnikova, O.: Multiword Expressions in NLP: General Survey and a Special Case of Verb-Noun Constructions. In: Emerging Applications of Natural Language Processing: Concepts and New Research. IGI Global. (2013) 1–21
8. Gelbukh, A., Kolesnikova, O.: Semantic Analysis of Verbal Collocations with Lexical Functions. Studies in Computational Intelligence 414 (2013) 146 pp.
9. Goldberg, Y., Orwant, J.: A Dataset of Syntactic-Ngrams over Time from a Very Large Corpus of English Books. Available: http://googleresearch.blogspot.mx/2013/05/syntactic-ngrams-over-time.html. Published May 23 (2013)
10. Ledeneva, Y., Gelbukh, A., García-Hernández, R.A.: Terms Derived from Frequent Sequences for Extractive Text Summarization. In: Proceedings of the International Conference on Intelligent Text Processing and Computational Linguistics, CICLing 2008. Lecture Notes in Computer Science 4919 (2008) 593–604
11. Pado, S., Lapata, M.: Dependency-based construction of semantic space models. Computational Linguistics, 33(2) (2007) 161–199
12. Padró, L., Collado, M., Reese, S., Lloberes, M., Castellón, I.: Freeling 2.1: Five years of open-source language processing tools. In: Proceedings of 7th Language Resources and Evaluation Conference (LREC 2010), ELRA (2010)
13. Sidorov, G: Syntactic Dependency Based N-grams in Rule Based Automatic English as Second Language Grammar Correction. International Journal of Computational Linguistics and Applications 4(2) (2013) 169–188.
14. Sidorov, G.: Non-continuous Syntactic N-grams. Polibits 48 (2013) 67–75 (in Spanish, abstract and examples in English).
15. Sidorov, G.: Non-linear construction of n-grams in computational linguistics: syntactic, filtered and generalized n-grams. (in Spanish) Mexico (2014) 166 pp.
16. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernandez, L.: Syntactic dependency-based n-grams as classification features. Lecture Notes in Artificial intelligence 7630 (2012) 1–11
17. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernandez, L.: Syntactic dependency-based n-grams: More evidence of usefulness in classification. In: Proceedings of the International Conference on Intelligent Text Processing and Computational Linguistics, CICLing 2013. Lecture Notes in Computer Science 7816 (2013) 13–24
18. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernandez, L. Syntactic N-grams as machine learning features for natural

language processing. Expert Systems with Applications 41(3) (2014) 853–860

GRIGORI SIDOROV
NATURAL LANGUAGE AND TEXT PROCESSING LABORATORY,
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN,
INSTITUTO POLITÉCNICO NACIONAL (IPN),
MEXICO CITY, MEXICO
WEB: <WWW.CIC.IPN.MX/~SIDOROV>