

A Python-based system for Verbalizing AMR Structures

Guy Lapalme
RALI-DIRO, Université de Montréal
lapalme@iro.umontreal.ca

Abstract

We describe a system for generating a literal reading of Abstract Meaning Representation (AMR) structures. The system uses a symbolic approach, in Python, to transform the original rooted graph into a tree of constituents that is transformed into an English sentence by an existing realizer. The system is quite fast and has been applied on a wide variety of AMR structures ($\approx 60K$ of them). The generated sentences are usually longer than the reference ones because they explicit all relations contained in the AMR. The majority of the generated sentences have been judged good or satisfactory for the stated goal: helping human annotators to check the AMRs they have written. A comparative evaluation with a recent AMT generator shows that although it does not generate a comparable linguistic quality, it is much faster to execute. Based on this experiment, we give suggestions for further AMR annotation.

This paper is a follow-up to a paper [15], written in 2019, that dealt with the same problem with a system called $\Gamma\omega\text{-}\Phi^1$ that used Prolog to create a constituent structure used by a JSREALB a Javascript based text realizer. Given the fact, that we have recently devised a set of Python classes to represent the constituent structures used by our realizer, we managed to reimplement the whole system in Python which is closely modeled on the previous Prolog architecture; this new system is called $\Gamma\omega\text{-}\Phi\pi^2$. We would probably never have designed the current implementation strategy, had we not developed it before in Prolog. We also present results on the new release of AMR (3.0) [14].

This paper is self-contained, it describes the Python implementation³ which realizes a similar output to that of the previous Prolog implementation. We report its results on the AMR (3.0) [14] recently released by LDC. We do not describe other AMR generators presented in our previous paper and although we show their output of a few sentences.

¹In homage to Donald Knuth, we used Greek letters for the name of the system that can be read aloud as GOPHI (Generation Of Parenthesized Human Input) pronounced as GOF AI (Generating Output From AMR Input) an acronym that also has an older reading originally given by John Haugeland.

²Continuing the convention of using greek letters for our system, which can be read as GOPHIPY (Generation Of Parenthesized Human Input in Python)

³Available at <https://github.com/lapalme/gophipy>

1 Introduction

The goal of this work is to create a system to *verbalize literally* Abstract Meaning Representation (AMR) [1] graph structures. An AMR represents the semantics of an English sentence by mapping different grammatical realizations into a single graph in order to focus on the meaning of the sentence. Important syntactic phenomena such as articles, number, tense and voice are not represented in the graph; variables being quantified existentially, universal quantification cannot be represented in all cases [6]. Given the current state of the art in NLP, parsing an English sentence in order to get its AMR graph is not yet reliable, so currently AMR graphs must be created by human annotators who fortunately can rely on useful tools such as a computer-aided editor [12]⁴ that validates the syntactic form of the graph and provide other annotating guides. These graphs are then revised in order to obtain consensus AMRs. One stated goal of the AMR project is to develop a large *semlbank* for shared tasks in natural language understanding and generation. The *AMR Bibliography* [4] gives a comprehensive list of recent works making use of AMR for NLP tasks.

Over the years, research groups have created AMRs for different types of texts: novels (e.g. Le Petit Prince), scientific texts in biology, news articles, English translations of Chinese texts, etc. The latest release by LDC (3.0) [14] provides $\approx 60\text{K}$ sentences with their AMR structures. This corpus has been used for developing automatic AMR parsers (sentence to AMR) and generators (AMR to sentence) by means of machine learning or machine translation techniques.

Their results were evaluated in the context of two *SemEval* tasks. *Task 8* of *SemEval-2016* [17] was devoted to the parsing of English sentences to get the corresponding AMR. *Task 9* of *SemEval-2017* [18] comprised two tasks: parsing biomedical sentences to get AMRs and generate English sentences from valid AMRs. A CoNLL 2020 Shared Task: *Cross-Framework Meaning Representation Parsing* [20] included AMR as one of the five graph-structured representations of sentence meaning that participants had to create from sentences.

Given the many mappings between a sentence and an AMR structure, automatically evaluating the output of such systems is quite difficult. For an AMR parser, approximate graph matching between the original graph and the one created by the parser is used [8]. For an AMR generator, BLEU scores [21] are used to compare the output sentence with the reference one.

All participants in the generation task used machine learning and statistical techniques. But as the input AMR is a formal language that can be easily parsed using a context-free grammar, we decided to try a symbolic approach. In 2019, we had developed $\Gamma\omega\text{-}\Phi$ that used Prolog for transforming the AMR into a constituent syntactic form⁵. $\Gamma\omega\text{-}\Phi$ was quite competitive with previous generators both in terms of quality of the generated sentence and execution speed. But it must be emphasized that the goal of our AMR generator is different from the previous attempts: we do not try to reproduce the reference sentence

⁴<https://amr.isi.edu/editor.html>

⁵ $\Gamma\omega\text{-}\Phi$ is available on the web at <http://rali.iro.umontreal.ca/amr/current/build/amrVerbalizer.cgi>

verbatim, but instead we generate a literal reading of the graph that we hope would be helpful for annotators when they create their graph by providing them a quick feedback on the annotation they have created.

SPRING [3] is a recent *game-changer* in the area of AMR generation whose results are significantly better than all previous generators, at least when comparing the BLEU scores (more than 11 BLEU points than the previous systems). It performs both AMR parsing and AMR generation with the same architecture, but here we focus on the generation aspect. It is built on BART [16], a pretrained Transformer encoder-decoder model through denoising for reconstructing an English text corrupted through shuffling, masking and sentence permutation. For generation, the linearization of the AMR is considered as a reordered, partially corrupted English sentence which has to be reconstructed. As BART is optimized for dealing with English words, the authors expand the tokenization vocabulary of BART by adding frequent relations names and roles. One drawback of this approach is the training time needing more than 16 hours on a GPU equipped computer; once trained, generation on a similar computer stills need minutes of computation.

In Section 5, we will compare our results with those of the *AMR-to-text* module of SPRING⁶ that we ran on a subset of the test set. The output of SPRING is also shown on a few examples together with the text produced by previous generators showing that it is in a class of its own. Given the excellent linguistic quality of the texts generated by SPRING compared to the one by $\Gamma\omega\text{-}\Phi\pi$ we did not feel that it was worth comparing these systems on this aspect. Instead, we manually evaluated a sample of 150 AMRs to measure to what extent the meaning of the original AMR is conveyed appropriately.

The wide gap between the reference and the corresponding graph is illustrated in the example of Table 1 in which the reference sentence is quite cryptic for people whose mother tongue is not English (like me) and who are not aware that *DH* refers to a cherished person and that *A&E* is a hospital department⁷. Such colloquial expressions whose semantics is *expanded* in the AMR are one of the reasons why BLEU scores are not very useful for improving the generation systems.

We do not consider the generated verbalization of $\Gamma\omega\text{-}\Phi\pi$ as perfect, but more explicit as it better describes all the elements of the original graph. In fact, during the development of our system, we managed to find a few typos and errors in some examples of the AMR Guidelines [2] which have been since corrected. This would probably not have happened using machine learning techniques which do not usually challenge input-output pairs.

Another motivation for our work was developing a *test bench* for JSREALB [19] (a bilingual French-English⁸ realizer written in Javascript) that has been developed in recent years in our lab⁹. As this realizer takes care of the details of the English language generated from an abstract constituency structure, we hoped it would be easier to go from an AMR to a JSREALB structure than to a well-formed English sentence.

⁶Available at github.com/SapienzaNLP/spring

⁷Surprisingly, SPRING manages to reproduce those abbreviations which are not given in the input, so we conjecture that this sentence had been previously encountered in the training set

⁸In this work, we only use the English realization part of JSREALB

⁹JSREALB is freely available at <https://github.com/rali-udem/jsRealB>

<pre> (n / need-01 :ARGO (p / person :ARGO-of (h / have-rel-role-91 :ARG1 (i / i) :ARG2 (h2 / husband)) :mod (d / dear)) :ARG1 (t / treat-03 :ARG1 p :location (d2 / department :topic (a / and :op1 (a2 / accident) :op2 (e / emergency)))) :time (a3 / after :op1 (a4 / attack-01 :ARG1 p))) </pre>	
<i>Reference</i>	DH needed treatment at A&E after the attack
$\Gamma\omega\text{-}\Phi\pi$	My dear husband needs to treat him in the department about the accident and the emergency after the attack.
SPRING	DH needed treatment at A and E after the attack.
JAMR	dear i have-rel-role husband person need to accident and emergency department treatment after attacks
ISI-MT	after the attack dear husband need treatment in an accident and emergency department
<i>Generate</i>	Dear person that have-rel-role i husband need person treat in department about accident , and emergency after person attack .
<i>Baseline</i>	person have-rel-role i husband dear need treat department and accident emergency after attack

Table 1: An AMR corresponding to the *Reference* sentence and their realization produced by different systems: $\Gamma\omega\text{-}\Phi\pi$ is the system described in this paper; JAMR and ISI-MT are existing generators based on machine learning techniques; *Generate* is the output **generate** button of the *AMR editor* [12] but as the author acknowledges, it is not very reliable because it has been developed in three days; *Baseline* is a generator written in twenty lines of Python described in Section 8.1.

It would be interesting to develop a machine-learning approach for transforming between these two formalisms, but we leave this as an *exercise to the reader*.

In the following, we first briefly recall what an AMR is and the constituency structure that we target. We then describe the intermediary representation that we use for transforming an AMR to an English sentence. The implementation of the system and the tests are then presented. The evaluation results (both automatic and manual) are given and compared with those produced by SPRING. We end with a discussion of the pros and cons of our approach. We also make suggestions for streamlining the AMR and for limiting the number of *primitives*. Three appendices give supplementary information: 8.1 compares the output of different AMR generators on a few selected AMRs; 8.2 gives implementation details for the core algorithm.

2 AMR concepts

An AMR is a singly rooted, directed acyclic graph with labels on edges (relations) and on nodes (concepts). AMR structures can use PROPBANK semantic roles [5], within-sentence coreference and can take into account some types of polarity and modality. The AMR Specifications [2] are the authoritative source for details about the formalism and its use for annotation of sentences.

The Lisp-inspired syntax of an AMR is quite simple: an AMR is either a variable, a slash and the name of a concept followed by a list, possibly empty, of role names followed by an AMR all within parentheses; an AMR can also be a variable reference or a string constant. A *concept* that stands for an event, a property or a state often corresponds to a PROPBANK frame or is an English noun, adjective or pronoun. A *role* name, an identifier preceded by a colon, indicates a relation between concepts; around 50 role names are predefined. A *variable* is a letter optionally followed by digits associated with a concept. The variable can be used to cross-reference a concept in an AMR. A *constant* is either a number, a quoted string or a plus or minus sign.

In the following, we will use the example of Table 2¹⁰ to illustrate the steps of our system. It is a simple AMR structure corresponding to the sentence **The boy desires the girl who does not like him**. As AMR structures abstract many common syntactic concepts such as number, tense or modality, it could also represent **The desire of boys for girls that do not like them** that we give as reference sentence.

The root of an AMR usually represent the focus of the sentence. This AMR contains two PROPBANK predicates (**desire-01** and **like-01**), each with two arguments (**:ARG0** and **:ARG1**) that stand respectively for the subject and the object of the verbs. **The boy** is the subject of **desire-01** and the object of **like-01**; in the former, it is referred to by the use of the variable **b** introduced above. Inverse roles, denoted with the suffix **-of**, can also be used to link two predicates to a single concept by keeping the focus: in our example, **girl** is the object of **desire-01** and object of **like-01**. Negation is indicated with the **:polarity** role

¹⁰This AMR is adapted from an example of the AMR Guidelines [2]

<pre>(d / desire-01 :ARG0 (b/boy) :ARG1 (g/girl :ARG0-of (l/like-01 :polarity - :ARG1 b)))</pre>	
<i>Reference</i>	The desire of boys for girls that do not like them
$\Gamma\omega\text{-}\Phi\pi$	The boy desires the girl who does not like him.
SPRING	The boy has a desire for a girl who doesn't like him.
JAMR	boy - like desire to girls
ISI-MT	the boy has a desire to be the girl who do not like
<i>Generate</i>	Boy desire girl that not like .
<i>Baseline</i>	boy desire girl that not like

Table 2: A simple AMR and their realization with the systems described in Table 1. The transformation steps followed by $\Gamma\omega\text{-}\Phi\pi$ to produce the English sentence are given Table 3.

whose value is - (minus). The table also shows the output produced by other generators for comparison. Table 3 illustrate the representations used in the transformation process:

Abstract Meaning Representation row 1 is the AMR shown in Table 2 with a graph showing relations between concepts;

Semantic Representation row 2 is a pretty-printed representation of the original AMR. The string representing the AMR is processed with a recursive-descent parser and saved as a Python expression built using classes. An AMR is displayed as a four elements within parentheses: the instance name, the concept, the list of roles possibly empty, and the instance name of the parent, preceded here by @. When an instance name refers to another AMR, it is preceded by ^. Inverse roles are *expanded* with a role starting with a star: e.g. `:*ARGO` replace `:ARGO` which itself is added as a role in the referenced concept whose instance refers to the original concept; the replacement eases the generation of a subordinate clause which keeps the focus on the last argument, the *girl* in our example;

Syntactic Representation row 3 is the result of transforming the Semantic Representation into a tree of constituents which corresponds to the input format that JSREALB uses to produce a well-formed English sentence.

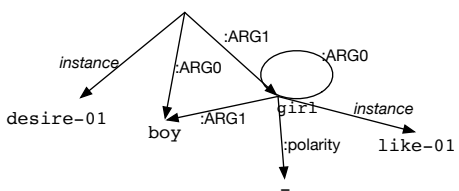
1	Abstract Meaning Representation	<pre>(d / desire-01 :ARG0 (b/boy) :ARG1 (g/girl :ARG0-of (l/like-01 :polarity - :ARG1 b)))</pre> 
2	Semantic Representation	<pre>(d desire-01 [:ARG0 (b boy [] @d), :ARG1 (g girl [::*ARG0 (l like-01 [:polarity (- # [] @1), :ARG1 (~b # [] @1), :ARG0 (~g # [] @1)] @g)] @d)])</pre>
3	Syntactic Representation	<pre>S(NP(D("the"), N("boy")), VP(V("desire"), NP(D("the"), N("girl"), SP(Pro("who"), S(VP(V("like"), Pro("me").pe(3).g("m"))).typ({"neg": true}))))))</pre>
4	English	The boy desires the girl who does not like him.

Table 3: Representations used in the transformation of the AMR structure shown in row 1 to the sentence shown in the last row. Row 2 is a representation of an internal Python representation of the AMR once it is parsed. Row 3 is a Python expression (which happens to be also a legal JavaScript expression) that is the serialization of the internal representation created by the transformation process of $\Gamma\omega\text{-}\Phi\pi$. This expression is used as input to JSREALB to realize the English sentence shown in row 4.

3 From Semantic Representation to Syntactic Representation

As getting the Semantic Representation from an AMR is a straightforward parsing job performed by recursive descent, the challenge is to transform the Semantic Representation to the Syntactic Representation (i.e. from the second to the third row of Table 3) which is detailed in this section.

The core of the system is conceptually simple: a Semantic Representation (SemR) is transformed compositionally by means of lambda expression applications. Each concept in the dictionary is encoded as a lambda expression that returns an Syntactic Representation (SyntR). When concept is applied to its arguments (i.e. its roles) it creates a new SyntR corresponding to their composition.

We illustrate this process with a simplistic example with the following Python definitions in which symbols starting with a capital letter are calls to a Python class constructor to create an internal Python structure corresponding to terminals and non-terminals in the classic constituent grammar notation. The constructors ignore None parameters.

```
like_01 = lambda arg0, arg1: S(arg0, VP(V("like"), arg1))
boy      = lambda d, a: NP(optD(d), a, N("boy"))
girl     = lambda d, a: NP(optD(d), a, N("girl"))
```

like_01 is a lambda expression that corresponds to a constituency tree with S as root; the first child is the subject given by arg0, the second child is a VP tree with its first child being the verb like and arg1 its second child, the object of the verb. When a lambda term is applied, a missing subject or object is indicated with the None value.

boy and girl are lambda expressions that build a constituency tree having NP as root with three children: the determiner, an adjective and the noun itself. To take into account optional parameters, we define the following function

```
optD = lambda det : det if det!=None else D("the")
```

which inserts the definite determiner the when None is specified.

Calling

```
like_01(boy(None, None), girl(D("a"), None))
creates a structure that can be pretty-printed as:
```

```
S(NP(D("the"),
    N("boy")),
  VP(V("like"),
    NP(D("a"),
      N("girl"))))
```

When an argument of a verb (e.g. give) should be realized with a preposition, the lambda associated with the verb is defined as follows where pp embeds the arguments in a prepositional phrase when it is not None:

```
give_01 = lambda arg0, arg1, arg2: S(arg0, VP(V("give"), arg1, pp("to", arg2)))
pp       = lambda prep, arg: PP(P(pre), arg) if arg!=None else None
```

Section 8.2 gives more information about the Python implementation of this process.

In AMR, argument values are given by the values of the roles which are recursively evaluated until they are given simple values found in the dictionary. AMR concepts being the ones of PROPBANK, we created 7,912 verb entries of our dictionary by parsing the XML structures of PROPBANK frames to determine the arguments and used the annotated examples for determining the proper preposition to use. We also parsed other PROPBANK XML entries for determining related 2,879 nouns and 1,352 adjectives. We also added about 33,300 nouns, adjectives, conjunctions and adverbs from an internal dictionary of our lab that did not appear in the entries of PROPBANK. Some pronouns and other parts of speech were added manually.

If AMR expressions were limited to PROPBANK concepts with frame arguments (i.e. `:ARGi`) as roles then we could limit ourselves to this single application process, but there are many other AMR peculiarities to take into account. We leave it to the reader to decide if these singularities are fundamental or just an artifact to ease annotation. Even then, we conjecture that a single and more uniform process would be preferable. We now describe how some of these special cases are handled. $\Gamma\omega\text{-}\Phi$ was developed on the AMR 2.0 corpus and we had hoped that some of these special cases could be limited in version 3.0. Unfortunately, it proved to be the inverse, many more *constructions* were introduced in this version.

3.1 Polarity

Negation in AMR is indicated using a special role (`:polarity`) associated with `-`; this does not fit well with the lambda application process that we explained above. Fortunately, JSREALB also uses this type of *flag* to indicate that the current sentence should be negated. This process was borrowed from a similar mechanism in SimpleNLG [10] in which it is possible to mark a sentence as negated; the realization process then modifies the dependency structure to produce the negation of the original sentence. In JSREALB, negation is indicated by specifying the `type` of sentence by adding `.typ("neg":true)` after the `S` constructor; this type of modification is called an `option` in JSREALB. An example of this is shown in the row 3 of Table 3. So when the application process encounters a `:polarity` role, it must keep track of the fact that an option should be added to this effect in the **Syntactic Representation**. This option *trick* is also used in other cases, for example for generating interrogative sentences or adding modals. In some AMRs, the `:polarity` role is also added to concepts that correspond to nouns or adjectives. Unfortunately this is not implemented by JSREALB, so we had to check for these special cases.

3.2 Inverse Roles

Inverse roles (indicated by `-of` at the end of the role name) also do not easily fit in the lambda application process. They are introduced in AMR mainly for keeping the focus on a single concept that is used in different frames. In our example of Table 3, `girl` is object in one frame and subject in another. Inverse roles can be quite delicate to process in the general case, but for our verbalization context we decided to systematically introduce a relative clause. After trying some alternatives, we resorted to a *hack*: we transform the original

AMR containing inverse roles into one that only uses active roles but flagged to indicate to the realizer that a relative clause should be introduced. Table 3 shows an example of this transformation between rows 1 and 2 where a `:ARGO-of` has been transformed into a `::ARGO` role with an explicit `:ARGO` role added in the inner concept. This approach is appropriate for *short* AMRs, but it often results in embedded relatives for *deep* ones that are almost unreadable.

3.3 Non-Core Roles

As described in the AMR Guidelines [2], there are 46 other *non-core* roles such as: `:domain`, `:mod` etc., each of which must be dealt specifically. But the basic principle remains the same, each AMR associated with a role is evaluated recursively and its **Syntactic Representation** is added to the deep structure of the current concept most often at the end of the values of the core roles. For example, for a `:destination`, we add a prepositional phrase starting by preposition `to`. In other cases, e.g. `:polite +`, then the word `Please` is added at the start of the syntactic structure. In an AMR, the `:wiki` role is used to disambiguate named entities, we use its value as the target of an HTML link that the system generates.

3.4 Special concepts

Some *special* concepts appear quite frequently and many new ones were added in the release 3.0 of AMR, but we did not manage to deal with them using the above framework.

`amr-unknown` is most often used for annotating an interrogative form; although intuitively this is convenient for the annotator and easy for a human to understand, automatically finding the appropriate form of interrogation is quite difficult because it depends on the enclosing role: e.g. if it appears as the value of an `:ARGO` or `:ARG1` then the question should start with `who`, if it appears as an argument of `:polarity`, then it should be a yes-or-no question, etc. Fortunately, JSREALB has options to transform affirmative into interrogative forms of different types, so in most cases, it is only a matter of adding the right option depending on the context. But there are still cases that are not dealt with correctly in the system. A closer examination of the use of `amr-unknown` revealed that this *vague* concept seemed to have been interpreted in diverse ways by different annotators.

`date-entity` has more than ten specific roles which have to be taken into account to represent time, date, day-period and even calendar type.

`government-organization` encodes the name of different governmental entities that should be realized in specific ways to match their official names.

`have-degree-91` indicates the comparison between two entities and often induces dependencies between structures of different roles. For example, a comparative is annotated

using different roles for the domain, the attribute, the degree, the object of the comparison and a possible reference to a superset. Finding appropriate verbalizations for all these cases proved quite tricky but essential as this concept is often used.

`have-polarity-91` is a reification (see Section 3.7) of polarity with a specific use of `:ARG2`.

`have-quant-91` which serves to mark a relation between an owner and specific types of quantifiable goods for certain goals.

`have-rel-role-91` indicates the relation between two entities (often child concept of `person`, another special concept) and the type of relation (e.g. `father`, `sister`,...) all using different roles. This also needs specific processing to get colloquial reading. For example, `my wife` is encoded as

```
(p / person :ARG0-of (h / have-rel-role-91 :ARG1 (i / i) :ARG2
  (w2 / wife)))
```

otherwise it would be verbalized as `the person who is relation of wife with me`.

`hyperlink-91` is used to connect a URL to regular text with specific roles values.

`multisentence` combines annotations and linking instances between many sentences. Verbalizing them is a simple matter of creating as many sentences as there are roles.

`ordinal-entity` to deal with some specific ways of verbalizing ordinal numbers: e.g. `last` for -1, `second to last` for -2, etc.

`*-quantity` there are about a dozen types of quantities, for which the roles `:quant` and `:unit` must be combined appropriately (using the value of the quantity as determiner for the unit when this value is a numeric value).

We did not manage to find a systematic handling of these special cases, we are left under the impression that they are a symptom of some design deficiencies in the original AMR formalism.

3.5 Pronoun generation

Variables in AMR enable the coreference between entities and create a graph between elements that could otherwise be considered as a tree. Although quite intuitive for the annotator, it proved to be quite intricate to generate the appropriate pronoun. Although we can deal properly with usual cases, there are still pending problems. Some of them have to do with the fact that, by design, AMR abstract important grammatical information such as gender and number; for example, *Steffi Graf* is referred to using `he` or *Yankees* is used as singular (see Figure 1). English pronouns also are different when used as nominative (`:ARG0` → `I`) or as accusative (`:ARG1` → `me`). But this *trick* does not always work because it depends on the argument structure of the concept and the fact that the subject or object is animate or not, an information that is not available because it would imply understanding the comments in the PROPBANK file.

3.6 Passive and other peculiarities

AMR does not indicate if the sentence should be realized as a passive. Instead annotators seem to rely on a convention that a verb with a subject, usually indicated by `:ARG0`, is used with an `:ARG1` instead. This fact must be checked before any other processing because passive is used extensively in English, especially in the news genre often encountered in the annotated corpora. Other special cases that must be checked are imperative and *yes or no question* that use the `amr-choice` special concept as `:ARG1`. Moreover the AMR editor provide some *shortcuts* that are expanded in the resulting AMR. For example, `cause-01` is expanded as an inverse role, that we have to check and *un-expand* to produce a more readable sentence. Many named entities¹¹ followed by a proper name are explicitly given in the AMR, but only the proper noun is written out in the text (see *Yankees* in Figure 1). $\Gamma\omega\text{-}\Phi\pi$ checks for many of these in order to simplify the output.

3.7 Verbalization and Reification

AMR is oblivious to verbalization, see our example of Table 2 for which *The boy desires* and *The desire of the boys* are annotated using the single concept `desire-01`. Using tables provided on the ISI website¹² for helping annotators, we added some verbalization information to the dictionary which is used to nominalize a verb when it does not have a subject.

Roles can sometimes be used as a concept, a process called *reification* e.g. `:location` is reified as `be-located-at-91`. This is used to indicate that the focus of the sentence is the locating process itself instead of the object being located or to the location itself. Mapping tables between roles and their reifications are given on the ISI website and we adapted them for our context. Our implementation *dereifies* the cases identified by Goodman [11] before processing the AMR.

3.8 Unknown Role or Concept

When an unknown concept is encountered, we use the fact that AMR is English centric and that English morphology is relatively simple, at least compared to French and German. So we merely use the name of the concept, after removing dashes or the frame number, as the word to add to the sentence. When an unknown role is encountered, the corresponding AMR value is added at the end of the current constituent, ignoring the name of the role.

3.9 Conclusion

Although the basic principle for creating an AMR verbalizator is a simple β -reduction of lambda forms, there are (too ?) many special cases that do not fit well within the declarative approach to the transformation from *Semantic Representation* to the *Syntactic Representation*.

¹¹A list is given at <https://www.isi.edu/~ulf/amr/lib/ne-types.html>

¹²Resource list section at <https://amr.isi.edu/download.html>

An important limitation of our current implementation is the fact that we do not take lexical ambiguity into account, so if a concept corresponds in the dictionary either to a verb or to a noun, we consider only the verb entry. Some AMR examples (e.g. given in [7]) include the part of speech in the name of the concept e.g. `check.v.01` or `check.n.01` instead of `check-01`, but our corpora do not provide this information.

4 Implementation

The whole system is written in Python (4,000 lines), of which more than 1,000 for dealing with *special cases* described in sections 3.1 to 3.4. The parsing of the PROPBANK XML files generates a 58,000 lines dictionary. The final realization of the English sentence is produced by JSREALB as a NODE.JS module taking as input the Syntactic Representation.

To develop $\Gamma\omega\text{-}\Phi\pi$, we used the same examples used for developing $\Gamma\omega\text{-}\Phi$: 268 examples from the AMR Guidelines [2] and the 826 examples from the AMR Dictionary [13]. These AMRs are usually short sentences that combine a few concepts for didactic purposes and are designed to cover most of the annotation cases; they are thus ideal for developing a system, even though most sentences in the *real* corpora are much longer and combine diverse roles in a single AMR. Release 3.0 of AMR has introduced a subset of the corpus (8027 AMRs) annotated at the document level for coreference, implicit role reference, and bridging relations.

We recall that our goal is not to reproduce verbatim the original sentence, but to give a literal reading of the AMR in which all verbs are generated at the present tense and in the active voice; all nouns are singular with a definite determiner. Given the fact that the original sentences usually have a great variation in number, tense (some informal sentences are even encountered, see Table 1), it is expected that the BLEU scores between the reference and our *standardized* output will not be high. Moreover, $\Gamma\omega\text{-}\Phi\pi$ also generates an HTML link when a `:wiki` role is used in an AMR (see the bottom part of Figure 1).

$\Gamma\omega\text{-}\Phi\pi$ produced a sentence for all AMRs of our development, test and training examples: corpora available at the ISI website¹³ and the ones from the AMR 3.0 Distribution [14] $\Gamma\omega\text{-}\Phi\pi$ run on a MacBook laptop (1.2 GHz without GPU!) is quite fast: a few milliseconds of CPU and about 7 ms of real time per sentence. Of course, longer sentences take more time to verbalize, but it is still quite fast compared to other statistical systems which take seconds for computing an English sentence even after a longer initial loading phase. These timings are for Python to parse the AMR and producing the Syntactic Representation, JSREALB is also almost instantaneous. The generated sentences are systematically longer than the reference sentence, between 15% up to 70% longer, for a mean of 38%.

We also developed a web server¹⁴, also written in Python, that displays a web page (see Figure 1) in which a user can edit an AMR. The AMR is then transformed and its English realization is generated by an instance of JSREALB integrated in the response web page itself. This setup allows the web links generated by JSREALB to be clicked directly

¹³<https://amr.isi.edu/download.html>

¹⁴<http://rali.iro.umontreal.ca/amr/python/current/export/cgi-bin/amrVerbalizer.cgi>

from this web page (see the words *Yankees* in the right part of the figure). This example (numbered isi.0001.12) taken from the AMR Dictionary gives as a reference sentence: *The boy doesn't think the Yankees will win.*¹⁵ in which the negation is not given the same scope in the reference and in the AMR. This is a case in which an AMR verbalizer could have helped catch this type of discrepancy because the annotator could have noticed that the generated sentence from the AMR has a slightly different meaning than the original sentence, so it should be double-checked.

Figure 1: Web interface to the verbalizer. The input page on the left shows an editor with a special *mode* for editing AMRs; intermediate representations can also be requested with the checkboxes at the bottom, only SyntR is chosen here. The right part shows the corresponding SyntR (in indented form) and the realized sentence created by the embedded JSREALB module in the webpage.

¹⁵This sentence is reproduced verbatim by SPRING.

5 Evaluation

AMR generation outputs up to now have been evaluated either by means of BLEU scores [9, 22, 3] and by means of pairwise comparisons [18]. But unfortunately these metrics are not very useful for helping the development or the improvement of a symbolic system.

5.1 Development Evaluation

So in order to keep track of the *progress* of our system, we devised a crude metric using the scale shown in Table 4. During the course of our development, we manually evaluated a small subset to measure the adequacy of the generated sentence for the intended purpose, that is, helping an annotator to check if the proper concepts and roles have been chosen in the annotation. This evaluation scale helped us focus on the most frequent drawbacks of the system.

- 4 Perfect translation of all concepts of the AMR and acceptable English formulation
- 3 Translation correct, but bad English formulation
- 2 Translation correct, but English barely understandable
- 1 Gibberish in the English
or missing important information from the AMR or bad meaning conveyed
- 0 Error in the parsing, translation or generation (not encountered anymore!)

Table 4: Scale used for the evaluation of the results

We do not give here the results on the development corpora, but instead we focus on the results obtained on a random sample 25 AMRs taken from the six test sets of the AMR 3.0 Distribution [14]. In order to focus on *interesting* AMRs, we removed those having less than 5 lines; They account for about 20% of all cases and are often only auxiliary informations such as dates, amounts, author names, etc. In doing so, we penalize the scores of the systems which almost always realize these **short** AMRs perfectly. We feel that the generation challenge sits in the realization of complex AMRs, otherwise a plain formatter would be sufficient.

Table 5 shows the score we gave for the both $\Gamma\omega\text{-}\Phi\pi$ and SPRING [3]. As the generation of web links is systematic, HTML tags were removed from the generated text before the evaluation. Comparison with the reference sentence is not taken into account in this evaluation as the generated verbs are always conjugated at the present tense, the nouns are always singular and the determiners always definite.

SPRING is the clear winner not only in terms of BLEU scores, but also in the number of perfect translations: in fact, in some cases, the SPRING formulations were more fluent than the reference one. 82% of the formulations produced by $\Gamma\omega\text{-}\Phi\pi$ and 87% by SPRING were considered as useful (scores 3 and 4). 12% of the $\Gamma\omega\text{-}\Phi\pi$ formulations had a very bad English formulation namely because it is strictly compositional and thus it creates complex sentences with many subordinates that mirror the nested nature of the AMR. In some cases, this makes it very hard to see what parts of a sentence relate to another.

Our crude scale assign 1 for two different reasons: as $\Gamma\omega\text{-}\Phi\pi$ explicits the input AMR, it never forgets to render any information, unless there is a bug in the system, so this score is only given for very bad formulation.

System	BLEU	CPU secs	Mean	4	3	2	1	0
$\Gamma\omega\text{-}\Phi\pi$	7.00	1.7	3.14	54	70	19	7	0
SPRING	34.62	264.0	3.60	128	3	0	19	0

Table 5: Statistics of the manual evaluation scores on a sample of 25 *long* AMRs taken from each the six split/test corpora of Release 3.0 of AMR for $\Gamma\omega\text{-}\Phi\pi$ and SPRING. The second column gives the BLEU scores. The third column gives the mean score over all 150 manual evaluations of this corpus; columns 4 to 8 give the number of sentences that were given the corresponding score between 4 and 0.

As this bad formulation problem was never encountered for SPRING, this score was given in 12% of cases when some information was missing or when the sentence did not convey appropriately the information found in the AMR input. Here is more detailed analysis of the these cases which are relatively hard to spot because the generated text is so fluent that one must be very careful in checking these translations:

- 6 cases of missing information such as number, dimension and even a case of missing `not`;
- creation of *strange* words: *prostitiators*, *go-go*;
- 9 cases of change in meaning: e.g. *China is stronger than Korea*, but the AMR says the inverse
- in the case of multi-sentence, the `snt i:` roles are sometimes not sorted in the AMR, so SPRING follows the order of the input and not of the `snt i:`, this problem could probably be fixed during the AMR linearization step;

The only clear advantage of $\Gamma\omega\text{-}\Phi\pi$ over SPRING is in computation time: it takes less than two seconds on a commodity laptop while SPRING needs more than four minutes on high-power machine with a GPU and many gigabytes of memory.

6 Future Work

This paper has described a symbolic AMR verbalizer that shows that the approach is viable and fast. There is still work to do on many aspects.

$\Gamma\omega\text{-}\Phi\pi$ should take lexical ambiguity into account. Currently if the same character string can refer to a verb, a noun or an adjective such as `war` or `good`, the system chooses the first fit in the above order. When this is not appropriate, we manually removed the *bad* entry (e.g. the verb entry for `war` and the noun entry for `good`). This process gives acceptable results most of the time, but it should be revised using a better linguistically justified process.

Currently an AMR is verbalized as a single sentence, except in the case of a **multisentence**; thus sentences are sometimes long-winded and even repetitious. This is due to the depth-first traversal of the graph and also because relative sentences are used most of the time for inverse roles. The nominalization should be better integrated and could sometimes help in reducing the verbosity of the generated text. A better use of pronouns could also help shorten or split long sentences.

Although the system has been tested on all available AMR corpora, the implementation is still a bit shaky: it involves feeding the output of a Python system ($\Gamma\omega\text{-}\Phi\pi$) into JSREALB, a Javascript module. An error in the input of JSREALB must sometimes be linked back to the Python system (often an error in the generated dictionary). This process should be streamlined even though we could run many ten of thousands of examples without any problem.

It might also be interesting to revisit the starting point of the transformation: there has been extensive work for generating text from **First-Order Logic** which can be systematically generated from AMR. It would probably be worth a try to revisit this aspect that we gave up perhaps a bit too soon.

It would also be interesting to try to use machine learning techniques to develop the transformation rules between the **Semantic Representation** and **Syntactic Representation**, hoping that the process will learn how to cope with many of the special cases. It would probably also solve some of the *hallucination* problems encountered by SPRING.

6.1 Suggestions for AMR Developers

We also think that AMR developers should benefit from taking a *generative* view: we have observed a recent tendency to add new concepts and roles as a way to ease the annotation, but this makes the generation process more complex. This proliferation of new *primitives* also makes the use of machine learning techniques more difficult as there are only very few instances (often only one) of each in the training material, although SPRING manages to generate excellent texts most of the time. We suggest that the AMR developers try to limit the number of roles to the bare minimum as they have successfully done with the syntactic peculiarities. The annotators should also limit the use of inverse roles, not only because they are difficult to verbalize by our system \smile , but because they do not seem to add to the semantics, especially when inverse roles are composed (i.e. the inverse role of a concept that is itself used in an inverse role). The problem has even been exacerbated in Release 3.0 of the AMR, because the release notes claim that the *AMR deepening* is an important feature of this release with hundreds of new frames, some of them very specialized such as `read-between-lines-09` or `take-with-grain-of-salt-36`.

Although Bos has shown that the **meaning** of AMRs can be expressed in first-order logic, it would be interesting to develop a theory of *non-core roles* (e.g. why are some roles important or necessary ?) and their relations with language or other NLP applications. Such a theory would have been helpful for us when developing our system; for example, `:ARGi` roles have been studied for a long time and their meaning is relatively well defined, so their implementation for generation is relatively *clean*. Many other roles do not seem to

have well-defined semantics (e.g. `:mode`, `:manner`, `:time` which depends on the context of use, etc.), some relations (e.g. comparisons) are expressed in many different ways in the corpus. A much more systematic coding of AMRs would surely simplify the analysis of their meaning and help in their use in future NLP applications.

The current AMR corpus is interesting in its diversity: news articles, biology texts, novels, tutorial examples, informal (even vulgar!) examples and contrived texts that *linguists love*¹⁶. This shows that AMR can convey almost any kind of text, but then it makes it difficult for system developers to focus on specific aspects. So it might be interesting to annotate texts that will target specific application areas.

7 Conclusion

We have described $\Gamma\omega\text{-}\Phi\pi$ a text generator from AMR input. It is a proof of concept of a feasible symbolic approach to AMR generation that takes advantage of the fact that AMR is a structured input that can serve as a plan for the output text. We have shown the interest of generating constituency structures instead of full sentences that can be obtained systematically from them. We have also shown that our previous system JSREALB can be a useful intermediary especially for producing variations of structure.

Acknowledgements

We thank Fabrizio Gotti for many fruitful discussions and suggestions, for helping with the evaluation and for installing the ISI AMR to English generator on our servers. We thank Philippe Langlais who made detailed suggestions for improving the organization of the paper.

References

- [1] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186. Association for Computational Linguistics, 2013.
- [2] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation (AMR) 1.2.5 Specification. <https://github.com/amrisi/amr-guidelines/blob/master/amr.md>, Feb 2018.

¹⁶See example isi.0002.315: Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

- [3] Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. One SPRING to rule them both: Symmetric AMR semantic parsing and generation without a complex pipeline. In *Proceedings of AAAI*, 2021.
- [4] Austin Blodgett and Nathan Schneider. AMR Bibliography. <https://nert-nlp.github.io/AMR-Bibliography/>.
- [5] Claire Bonial, Julia Bonn, Kathryn Conger, Jena D. Hwang, and Martha Palmer. Prop-Bank: Semantics of new predicate types. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. European Language Resources Association (ELRA), 2014.
- [6] Johan Bos. Expressive power of Abstract Meaning Representations. *Comput. Linguist.*, 42(3):527–535, September 2016.
- [7] Johan Bos. Separating argument structure from logical structure in amr separating argument structure from logical structure in amr. arXiv:1908.01355v1, August 2019.
- [8] Shu Cai and Kevin Knight. Smatch: an Evaluation Metric for Semantic Feature Structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752. Association for Computational Linguistics, 2013.
- [9] Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime G. Carbonell. Generation from abstract meaning representation using tree transducers. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 731–739. The Association for Computational Linguistics, 2016.
- [10] Albert Gatt and Ehud Reiter. SimpleNLG: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 90–93. Association for Computational Linguistics, 2009.
- [11] Michael Wayne Goodman. AMR normalization for fairer evaluation. In *Proceedings of the 33rd Pacific Asia Conference on Language, Information, and Computation*, Hakodate, 2019.
- [12] Ulf Hermjakob. AMR Editor. <https://amr.isi.edu/editor.html>.
- [13] Ulf Hermjakob. AMR Annotation Dictionary. <https://www.isi.edu/~ulf/amr/lib/amr-dict.html>, May 2018.
- [14] Kevin Knight, Bianca Badarau, Laura Baranescu, Claire Bonial, Madalina Bardocz, Kira Griffitt, Ulf Hermjakob, Daniel Marcu, Martha Palmer, Tim O’Gorman, and Nathan Schneider. Abstract Meaning Representation (AMR) Annotation Release 3.0. <https://catalog.ldc.upenn.edu/LDC2020T02>, 2020.

- [15] Guy Lapalme. Verbalizing AMR structures. <http://rali.iro.umontreal.ca/rali/sites/default/files/publis/GoPhi.pdf>, 08/2019 2019.
- [16] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.
- [17] Jon May. SemEval-2017 Task 8: Meaning Representation Parsing. <http://alt.qcri.org/semeval2016/task8/>, 2016.
- [18] Jon May. SemEval-2017 Task 9: Abstract Meaning Representation parsing and generation. <http://alt.qcri.org/semeval2017/task9/>, 2017.
- [19] Paul Molins and Guy Lapalme. JSrealB: A bilingual text realizer for web programming. In *European Conference on Natural Language Generation (Demo)*, pages 109–111, Brighton, UK, sep 2015.
- [20] Stephan Oepen, Omri Abend, Lasha Abzianidze, Johan Bos, Jan Hajic, Daniel Herscovich, Bin Li, Tim O’Gorman, Nianwen Xue, and Daniel Zeman. MRP 2020: The second shared task on cross-framework and cross-lingual meaning representation parsing. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, pages 1–22, Online, November 2020. Association for Computational Linguistics.
- [21] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL ’02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.
- [22] Nima Pourdamghani, Kevin Knight, and Ulf Hermjakob. Generating English from Abstract Meaning Representations. In Amy Isard, Verena Rieser, and Dimitra Gkatzia, editors, *INLG 2016 - Proceedings of the Ninth International Natural Language Generation Conference, September 5-8, 2016, Edinburgh, UK*, pages 21–25. The Association for Computer Linguistics, 2016.

8 Appendix

8.1 Some AMRs with output produced by some generators

The following tables give a few AMR structures, the corresponding reference English sentences and the sentences produced by 6 different generators to give a perspective on the current state of our system and that of other generators:

$\Gamma\omega\text{-}\Phi\pi$ System described in this paper, we indicate in parentheses the development evaluation score given to this sentence;

SPRING System described by Bevilacqua et al. [3] and run using a Google Colaboratory machine according to the instructions given on the implementation site ¹⁷

JAMR Pretrained generation model of Flanigan [9] used *out of the github* with the provided Gigaword corpus 4-grams;

ISI-MT System developed by [22] and made available at the ISI website as the **AMR-to-English generator**¹⁸;

Generate Output of the **generate** button in the AMR Editor [12];

Baseline Our own baseline generator (20 lines of Python) that merely does a top-down recursive descent in the AMR structure and outputs the concepts encountered. If an **:ARGO** is present, it is output before the root concept in order to try to keep the subject in front of the verb. **not** is added when a negative polarity is encountered and **that** is inserted in the case of an inverse role.

¹⁷[github/SapienzaNLP/spring](https://github.com/SapienzaNLP/spring)

¹⁸https://www.isi.edu/projects/nlg/software_1

<pre>(f / fire-01 :ARG0 (a / aircraft-type :wiki "Mikoyan-Gurevich_MiG-25" :name (n / name :op1 "MiG-25")) :ARG1 (m / missile :source (a2 / air) :direction (a3 / air)) :destination (a4 / aircraft-type :wiki "General_Atomics_MQ-1_Predator" :name (n2 / name :op1 "Predator")))</pre>	
<i>Reference</i>	The MiG-25 fired an AAM at the Predator.
$\Gamma\omega\text{-}\Phi\pi$ (4)	 MiG-25 fires the missile from the air to the air to Predator
SPRING	The MiG-25 fired an AAM at the Predator.
JAMR	mig-25 mikoyan-gurevich_mig-25 general_atomics_mq-1_predator predator aircraft fired the missiles in the air in the air
ISI-MT	fire aircraft-type :wiki mikoyan-gurevich_mig mig-21 missiles from the air to air to aircraft-type :wiki general_atomics_mq-1_predator predator
<i>Generate</i>	fire missile air from air to
<i>Baseline</i>	aircraft-type "Mikoyan-Gurevich_MiG-25" name "MiG-25" fire missile air air aircraft-type "General_Atomics_MQ-1_Predator" name "Predator"

Table 6: AMR and their realization with different systems (ex: isi_0002.701)

<pre>(k / know-01 :polarity - :ARG0 (i / i) :ARG1 (t / truth-value :polarity-of (s / straight-05 :ARG1 (h / he))))</pre>	
<i>Reference</i>	IDK if he's str8.
$\Gamma\omega\text{-}\Phi\pi(4)$	I do not know whether he is straight.
SPRING	I don't know if he's str8.
JAMR	i know that he is straight -
ISI-MT	I not know truth-value that was noted straight he.
<i>Generate</i>	i don't know truth-value :polarity-of straight from him.
<i>Baseline</i>	i not know truth-value straight he

Table 7: AMR and their realization with different systems (ex:isi_0002.691)

<pre> (b / bind-01 :ARG1 (s / small-molecule :wiki - :name (n / name :op1 "TKI258")) :ARG2 (e / enzyme :wiki "Fibroblast_growth_factor_receptor_1" :name (n2 / name :op1 "FGFR1") :ARG1-of (k / knock-down-02) :ARG2-of (m / mutate-01 :value "V561M")) :ARG4 (b2 / binding-affinity-91 :ARG1 (i2 / inhibitor-constant) :ARG2 (a / approximately :op1 (c / concentration-quantity :quant 35 :unit (n3 / nanomolar)))) :ARG4 (t / tight-05)). </pre>	
<i>Reference</i>	TKI258 binds tightly to the FGFR1 KD V561M (Ki 35 nM)
$\Gamma\omega\text{-}\Phi\pi$ (3)	Is bound TKI258 FGFR1 that knocks down that mutate V561M with the binding-affinity inhibitor constant approximately nanomolar 35 tight.
SPRING	TKI258 binds tightly to the FGFR1 KD V561M (K₂ 35 nM).
JAMR	inhibitor-constant binding-affinity approximately 35 nanomolar tightly binds tki258 - with knock-down mutating v561m fgfr1 fibroblast_growth_factor_receptor_1
ISI-MT	bound small-molecule :wiki doesn't tki258 enzyme :wiki fibroblast_growth_factor_receptor_1 fgfr1 by knock-down of mutating v561m to binding-affinity inhibitor-constant approximately 35 concentration-quantity nanomolar to tight
<i>Generate</i>	bind that was knock-downed and was mutated tight .
<i>Baseline</i>	bind small-molecule - name "TKI258" enzyme "Fibroblast_growth_factor_receptor_1" name "FGFR1" knock-down mutate "V561M" binding-affinity inhibitor-constant approximately concentration-quantity 35 nanomolar tight

Table 8: An AMR for a biology text as generated by different systems (ex: isi_0002.786)

<pre> (k / know-01 :ARG0 (i / i) :ARG1 (t / thing :ARG0-of (c / cause-01 :ARG1 (c2 / cross-02 :ARG0 (c3 / chicken) :ARG1 (r / road)))))) </pre>	
<i>Reference</i>	I know why the chicken crossed the road.
$\Gamma\omega\text{-}\Phi\pi$ (4)	I know the thing that causes that the chicken crosses the road.
SPRING	I know why the chicken crossed the road.
JAMR	i know the things that cause the chicken cross the road
ISI-MT	know why the chicken cross the road?
<i>Generate</i>	I know thing that caused chicken crossing road .
<i>Baseline</i>	i know thing cause chicken cross road

Table 9: AMR and their realization with different systems (ex: isi_0002.766)

<pre> (a / and :op1 (r / remain-01 :ARG1 (c / country :wiki "Bosnia_and_Herzegovina" :name (n / name :op1 "Bosnia")) :ARG3 (d / divide-02 :ARG1 c :topic (e / ethnic))) :op2 (v / violence :time (m / match :mod (f2 / football) :ARG1-of (m2 / major-02)) :location (h / here) :frequency (o / occasional)) :time (f / follow-01 :ARG2 (w / war :time (d2 / date-interval :op1 (d3 / date-entity :year 1992) :op2 (d4 / date-entity :year 1995)))))) </pre>	
<i>Reference</i>	following the 1992-1995 war, bosnia remains ethnically divided and violence during major football matches occasionally occurs here.
$\Gamma\omega\text{-}\Phi\pi$ (3)	Bosnia remains under to divide it about ethnic, the occasional violence the football match that is major here and when follows the war from 1992 to 1995.
SPRING	Following the wars of 1992-1995 Bosnia remains ethnically divided and there has occasionally been violence in major football matches here.
JAMR	following the 1992 1995 war , ethnic divides bosnia bosnia_and_herzegovina remains , and the occasional major football match violence in here
ISI-MT	following the war between 1992 and 1995 countries :wiki bosnia_and_herzegovina bosnia remain divided on ethnic and violence at football matches by major here from time to time.
<i>Generate</i>	Bosnia remains Bosnia divided about ethnic , and violence in here football match that was majored following war from 1992 to 1995 .
<i>Baseline</i>	and remain country "Bosnia_and_Herzegovina" name "Bosnia" divide ethnic violence match football major here occasional follow war date-interval date-entity 1992 date-entity 1995
RIGOTRIO	following the 1992 1995 war, bosnia has remained an ethnic divide, and the major football matches occasionally violence in here.
CMU	following the 1992 1995 war , bosnia remains divided in ethnic and the occasional football match in major violence in here
FORGe	Bosnia and Herzegovina remains under about ethnic the Bosnia and Herzegovina divide and here at a majored match a violence.
ISI	following war between 1992 and 1995 , the country :wiki bosnia and herzegovina bosnia remain divided on ethnic and violence in football match by major here from time to time
Sheffield	Remain Bosnia ethnic divid following war 920000 950000 major match footbal occasional here violency

Table 10: Example used at SemEval-2017: Task 9; the last 5 lines show the output from participants given by the task organizers [18]

8.2 Python Implementation of Lambda Application

As our generation algorithm is based on lambda application (see Section 3), it might be interesting to some readers to see how this process is implemented in Python. As Python is a functional language, it already implements function application, so the only *challenge* left is setting up the proper environment for evaluation by linking the arguments with the SyntRs returned by the evaluation of the roles of an AMR.

Instead of associating a word with a simple lambda as we showed in Section 3, the lexicon associates each word with an instance of the `LexSem` class defined in Listing 1.¹⁹ It uses instance of the `Env` and `Option` classes (Listings 2 and 3) which respectively keep track of the values of the arguments and of the options that are to be added by JSREALB.

In the `LexSem` class, the *important* work starts at line 12 which builds a list of actual argument values by looking up in the environment if there exists a value for each parameter, if so it adds it to the list otherwise it inserts a `None` that will be ignored by the SyntR constructors. This argument vector is then applied to the lambda to build the SyntR to which the unprocessed arguments are added (not shown here). Finally, (line 14) options for the whole phrase are added.

```
1 class LexSem:
2     def __init__(self, lemma, pos, args, lambda_):
3         self.lemma=lemma      # useful for str(..)
4         self.pos=pos          # part of speech, useful for str(...)
5         self.args=args        # list of arguments as they appear in the AMR
6         self.lambda_=lambda_ # function associated with the word
7
8     def apply(self, env=None, opts=None):
9         if env==None: env=Env()
10        if opts==None: opts=Options()
11        ## process args from dictInfo building the list of arguments or
12        None
13        argV=[env.get(arg) if arg in env else None for arg in self.args]
14        syntR = self.lambda_(*argV)
15        return opts.apply(syntR)
```

Listing 1: Core of the lambda application in Python.

Listing 4 shows a few lexicon entries for noun and verbs with auxiliaries functions.

Listing 5 give a few examples of applications with the resulting SyntR displayed in comments along with the realized sentence by JSREALB.

¹⁹Some implementation details are not shown here

```

1 class Env:
2     def __init__(self,pairs=None):
3         self.pairs=pairs if pairs!=None else []
4
5     def __contains__(self,kind):
6         for rolei,_ in self.pairs:
7             if kind==rolei:return True
8         return False
9     def __getitem__(self,arg):
10        for key,val in self.pairs:
11            if key==arg:return val
12        return None
13
14    def put(self,arg,value):
15        self.pairs.append((arg,value))
16        return self
17
18    ## returns a list of elements associated with kind and remove them
19    from the environment
20    def get(self,kind):
21        res=[]
22        i=0
23        while i<len(self.pairs):
24            argi,rolei=self.pairs[i]
25            if kind==argi:
26                res.append(rolei)
27                self.pairs.pop(i)
28            else:
29                i+=1
30        return res

```

Listing 2: Environment representation.

```

1 class Options:
2     def __init__(self, opts=None):
3         self.opts=opts if opts!=None else []
4
5     def add(self, opt, value):
6         if opt in ["typ", "dOpt"]:
7             try:
8                 idx=self.opts.index(opt)
9                 self.opts[idx].update(value)
10            except ValueError :
11                self.opts.append((opt, value))
12        else:
13            self.opts.append((opt, value))
14        return self
15
16    def apply(self, syntR):
17        for opt, value in self.opts:
18            getattr(syntR, opt)(value)
19        return syntR

```

Listing 3: Options representation.

```

1 pp          = lambda prep, arg: PP(P(prep),arg) if arg!=None else None
2 optD       = lambda det : det if det!=None else D("the")
3
4 def showSyntR(syntR):
5     print(syntR.show()) # show the indented structure
6     print(jsRealB(syntR.show(-1))) # get realized string
7
8 verbs={}
9 nouns={}
10
11 verbs['give-01']=LexSem("V", "give", [":ARG0", ":ARG1", ":ARG2"],
12     lambda arg0, arg1, arg2:S(arg0, VP(V("give"), arg1, pp("to", arg2)))
13     )
14 nouns['envelope']= LexSem("envelope", "N", [":D", ":A"],
15     lambda d, a:NP(optD(d), a, N("envelope")))
16 nouns['boy']      = LexSem("boy", "N", [":D", ":A"],
17     lambda d, a:NP(optD(d), a, N("boy")))
18 nouns['girl']     = LexSem("girl", "N", [":D", ":A"],
19     lambda d, a:NP(optD(d), a, N("girl")))

```

Listing 4: Lexicon representation.

```

1 boyEnv=Env ([:D',D("a")])
2 boyEnv.put(":A",A("nice")).put(":A",A("little"))
3 boySyntR=nouns["boy"].apply(boyEnv,Options([("n","p")]))
4 showSyntR(boySyntR)
5 # NP(D("a"),
6 #   A("nice"),
7 #   A("little"),
8 #   N("boy")).n("p")
9 # nice little boys
10
11 envelope=nouns["envelope"]
12 envelopeSyntR=envelope.apply()
13 showSyntR(envelopeSyntR)
14 # NP(D("the"),
15 #   N("envelope"))
16 # the envelope
17
18 girl=nouns["girl"]
19 girlSyntR=girl.apply(Env([(:D",D("this"))]))
20 showSyntR(girlSyntR)
21 # NP(D("this"),
22 #   N("girl"))
23 # this girl
24
25 give=verbs["give-01"]
26 giveSyntR=give.apply(Env([(:ARG0",boySyntR),
27                           (:ARG1",envelopeSyntR),
28                           (:ARG2",girlSyntR)]),
29                   Options([("typ",{ "neg":True}]))))
30 showSyntR(giveSyntR)
31 # S(NP(D("a"),
32 #   A("nice"),
33 #   A("little"),
34 #   N("boy")).n("p"),
35 #   VP(V("give"),
36 #     NP(D("the"),
37 #       N("envelope")),
38 #     PP(P("to"),
39 #       NP(D("this"),
40 #         N("girl")))).typ({"neg": true})
41 # Nice little boys do not give the envelope to this girl.

```

Listing 5: Examples of lambda applications