RIGOTRIO at SemEval-2017 Task 9: Combining Machine Learning and Grammar Engineering for AMR Parsing and Generation

Normunds Gruzitis[1]**, Didzis Gosko**[2] **and Guntis Barzdins**[1]

[1]University of Latvia, IMCS / Rainis blvd. 29, Riga, Latvia
normunds.gruzitis@lumii.lv, guntis.barzdins@lumii.lv
[2]LETA / Marija street 2, Riga, Latvia
didzis.gosko@leta.lv

Abstract

By addressing both text-to-AMR parsing and AMR-to-text generation, SemEval-2017 Task 9 established AMR as a powerful semantic interlingua. We strengthen the interlingual aspect of AMR by applying the multilingual Grammatical Framework (GF) for AMR-to-text generation. Our current rule-based GF approach completely covered only 12.3% of the test AMRs, therefore we combined it with state-of-the-art JAMR Generator to see if the combination increases or decreases the overall performance. The combined system achieved the automatic BLEU score of 18.82 and the human Trueskill score of 107.2, to be compared to the plain JAMR Generator results. As for AMR parsing, we added NER extensions to our SemEval-2016 general-domain AMR parser to handle the biomedical genre, rich in organic compound names, achieving Smatch F1=54.0%.

1 Introduction

AMR (Banarescu et al., 2013) as a sentence-level semantic representation is evolving towards interlingua at SemEval-2017 Task 9 on Abstract Meaning Representation Parsing and Generation (May and Priyadarshi, 2017). The challenge was to improve over state-of-the-art systems for both text-to-AMR parsing (Barzdins and Gosko, 2016) and AMR-to-text generation (Flanigan et al., 2016).

AMR parsing subtask this year focused on specific genre of Biomedical scientific articles regarding cancer pathway discovery. Such texts are challenging to existing AMR parsers because they are rich in organic compound names with types “enzyme”, “aminoacid”, etc. not recognized by common NER tools that are often restricted to types “person”, “organization”, “location”, etc.

The paper starts with NER extensions used for the Biomedical AMR parsing subtask, followed by a novel approach of using Grammatical Framework for AMR generation, and concludes with a brief analysis of our SemEval results.

2 Text-to-AMR parsing

Only two adaptations to the AMR parser from SemEval-2016 (Barzdins and Gosko, 2016) were implemented: it was retrained on the union of LDC2015E86, LDC2016E25, LDC2016E33 and Bio AMR Corpus, and a gazetteer was added to extend the NER coverage to organic compound names found in the Bio AMR Corpus (e.g. “B-Raf enzyme”, “dabrafenib small-molecule”, etc.). The gazetteer was generalized w.r.t. numbers used in the names.

Although we achieved an above average Smatch score (54.0% versus 53.6%) in the preliminary official scoring, the ablation metrics show that we scored below average for named entities (46.0% versus 55.8%) and wikification (0% versus 33.0%). Since we used gazetteers extracted from the training data for both named entities and wikification, this suggests that external data sources should have been used instead.

3 AMR-to-text generation

Our approach to text generation from AMR graphs stems from a recent feasibility study (Gruzitis and Barzdins, 2016) on the grammar-based generation of storyline highlights – a list of events extracted from a set of related documents. The events would be represented by pruned AMR graphs acquired by an abstractive text summarizer and verbalized afterwards.

Such storyline highlight extraction is a part of the H2020 research project SUMMA, *Scalable Understanding of Multilingual MediA*.[1] The storyline highlights are expected to be relatively simple and concise in terms of grammatical structure and, thus, in terms of the underlying meaning representation. In our use case, adequacy and semantic accuracy of the generated sentences, and the control of the generation process are more important than fluency. Therefore we are following a grammar-based approach for AMR-to-text generation. In the SemEval task, however, we are pushing the limits and scalability of such approach, as the task requires much more robust wide-coverage general-purpose generation.

The proposed approach builds on Grammatical Framework, GF (Ranta, 2011). GF is a grammar formalism and technology for implementing computational multilingual grammars. GF grammars are bi-directional; however, they are particularly well suited for language generation. Most importantly, GF provides a wide-coverage resource grammar library with a language-independent API – a shared abstract syntax. The idea is to transform the AMR graphs to the GF abstract syntax trees (AST), leaving the surface realization (linearization) of ASTs to the existing English resource grammar.[2] Since the GF resource grammar library supports many more languages, this approach automatically extends to multilingual AMR-to-text generation, provided that there is a wide-coverage translation lexicon which includes named entities.

Because the coverage of our hand-crafted AMR-to-AST transformation rules is currently far from complete, we use the JAMR generator (Flanigan et al., 2016) as a default option for AMRs not fully covered by the rules. In other words, we want to measure if sentences generated by the GF approach (from AMRs fully covered by the transformation rules) outperform the respective JAMR-generated sentences. If so, it would be worth developing this approach further.

3.1 Grammatical Framework

More precisely, GF is a categorial grammar formalism and a functional programming language, specialized for multilingual grammar development. It has a command interpreter and a batch compiler, as well as Haskell and C run-time systems for parsing and linearization. The C run-time system has Java and Python bindings, and it allows for probabilistic parsing as well. Compiled GF grammars can be embedded in applications written in other programming languages.[3]

The key feature of GF grammars is the division between the abstract syntax and the concrete syntax. The abstract syntax defines the language-independent semantic structure and terms, while the concrete syntax defines the surface realization of the abstract syntax for a particular language. The same abstract syntax can be equipped with many concrete syntaxes (and lexicons) – reversible mappings from ASTs to feature structures and strings – making the grammar multilingual (Ranta, 2004).

What makes the development of GF application grammars rapid and flexible is the general-purpose GF resource grammar library, RGL (Ranta, 2009). The library currently covers more than 30 languages that implement the same abstract syntax, a shared syntactic API. The API provides overloaded constructors like

$$\mathrm{mkVP} : \mathrm{V2} \to \mathrm{NP} \to \mathrm{VP}$$
$$\mathrm{mkVP} : \mathrm{VP} \to \mathrm{Adv} \to \mathrm{VP}$$

for building a verb phrase from a transitive verb and an object noun phrase, or for attaching an adverbial phrase to a verb phrase, etc. – all without the need of specifying low-level details like inflectional paradigms, syntactic agreement and word order. These details are handled by the language-specific resource grammars.

Note that the overloaded API constructors generalize over the actual functions of the abstract syntax. The respective RGL functions of the above given mkVP constructors are

$$\mathrm{ComplV2} : \mathrm{V2} \to \mathrm{NP} \to \mathrm{VP}$$
$$\mathrm{AdvVP} : \mathrm{VP} \to \mathrm{Adv} \to \mathrm{VP}$$

These constructors and functions are applied to build ASTs. For instance, the sentence

“The boys want an adventure.”

is represented by the following AST w.r.t. RGL:

```
(PredVP
    (DetCN
        (DetQuant DefArt NumPl)
```

[1] http://summa-project.eu

[2] In terms of GF, linearization refers to resolving the word order, word forms (agreement), function words, etc.

[3] http://www.grammaticalframework.org

```
        (UseN boy_N))
    (ComplV2
        want_V2
        (DetCN
            (DetQuant IndefArt NumSg)
            (UseN adventure_N))))
```

The respective API constructor application tree is more general and simpler:

```
(mkCl
    (mkNP the_Quant pluralNum boy_N)
    (mkVP
        want_V2
        (mkNP a_Quant adventure_N)))
```

where want_V2, boy_N and adventure_N are nullary lexical functions, while the_Quant and a_Quant are predefined constructors for the function words, and pluralNum is a parameter for selecting the plural form of the noun. In GF, there is no formal distinction between syntactic and lexical functions.

We map AMRs to ASTs by a sequence of pattern-matching transformation rules, using RGL API constructors as a convenient intermediate layer. The language-specific linearization of the acquired ASTs is already defined by the English (or other language) resource grammar and lexicon.

3.2 AMR-to-AST transformation

The overall transformation process is as follows:

1. The input AMR is rewritten from the PENMAN notation to the the LISP-like bracketing tree syntax by a simple parsing expression grammar.
2. In case of a multi-sentence AMR, the graph is split into two or more graphs to be processed separately.
3. For each AMR graph represented as a LISP-like tree, a sequence of tree pattern-matching and transformation rules is applied, acquiring a fully or partially converted AST constructor application tree w.r.t. the API of RGL.
4. In case of a partially converted AST, the pending subtrees are just pruned.[4]
5. The resulting ASTs are passed to the GF interpreter for linearization.

[4]For the SemEval submission, we took the respective JAMR-generated sentence instead, skipping the fifth step.

Inspired by Butler (2016), we use the Stanford JavaNLP utilities Tregex and Tsurgeon (Levy and Andrew, 2006) for the pilot implementation of AMR-to-AST conversion.[5] The difference of our approach is that we convert AMR graphs to abstract instead of concrete syntax trees, and the choice of GF allows for further multilingual text generation, preserving grammatical and semantic accuracy.

In the time frame of the SemEval task, we defined around 200 transformation rules[6] covering many basic and advanced constructions used in AMR.[7] To illustrate the ruleset, let us consider the AMR graph given in Figure 1, and its expected AST in Figure 2, acquired by the ordered rules outlined in Figure 3. For each rule, P denotes a simplified Tregex pattern (a subtree to match), and R denotes the resulting subtree – after Tsurgeon operations like adjoin, move, relabel, delete have been applied (omitted in Figure 3). Note that we first slightly enrich the original AMR by adding frame-specific semantic roles to ARG2..ARGn. In our example, :ARG4 is rewritten to :ARG4-GOL, based on PropBank. Semantic roles are used to determine the preposition in a preposition phrase (see the ninth rule in Figure 3). Thus, we get GOL_Prep for the NP under :ARG4 of go-02, which, in the current prototype, is always linearized as the preposition "to". Although this preposition fits to our example, in general, other preposition can be used in the realization of GOL. More elaborated post-processing is needed to reconstruct the prepositions that are lost in the AMR representation of the input sentence. Statistics from the PropBank corpus (Palmer et al., 2005) would be helpful to decide whether there is a dominant frame-dependent preposition for the argument/role, or a dominant NP-dependent preposition, independently of the frame.

In addition to the regular AMR constructions, we have defined a number of rules for the treatment of the special frames: have-org-role-91 and have-rel-role-91. The special rules are applied before the regular ones. We have also introduced some post-editing rules over the resulting ASTs to

[5]`https://github.com/GrammaticalFramework/gf-contrib/tree/master/AMR/AMR-to-text`

[6]More precisely, we have defined 198 Tregex patterns over AMR/AST trees. For each pattern, 2.5 Tsurgeon transformation operations are defined on average (493 in total).

[7]Roughly estimating, the development of the current ruleset took us less than two person months.

```
(w / want-01
   :ARG0 (b / boy)
   :ARG1 (g / go-02
      :ARG0 b
      :ARG4 (c / city
         :name (n / name
            :op1 "New"
            :op2 "York"
            :op3 "City")
         :wiki "New_York_City")))
```

Figure 1: An AMR representing the sentence “The boys want to go to New York City”.

```
(mkText (mkUtt (mkS
 (mkCl
  (mkNP a_Quant (mkCN boy_N))
  (mkVP
   want_VV
    (mkVP
     (mkVP go_V)
     (mkAdv
      GOL_Prep
       (mkNP (mkPN "New York City")))))
 ))) fullStopPunct)
```

Figure 2: An AST acquired from the AMR given in Figure 1. When linearized, the AST yields “A boy wants to go to New York City” in English, or “En pojke vill gå till New York City” in Swedish.

make the final linearization more fluent. For instance, simple attributive relative clauses are converted to adjective modifiers; e.g. “luck that is good” gets converted to “good luck”. Similarly, it would be often possible to convert general nouns modified by simple verbal relative clauses into more specific nouns omitting the use of relative clauses: “person that reports” – to “reporter”, “organization that governs” – to “government”, etc.

Regarding the RGL lexicon, it contains more than 60,000 lexical entries, providing a good coverage for general purpose applications. To handle out-of-vocabulary words and multi-word expressions which most frequently are named entities (e.g. “New York City”), we use low-level RGL constructors to specify fixed strings.

3.3 JAMR Generator

A pre-trained JAMR generation model (Flanigan et al., 2016) along with provided Gigaword corpus 4-grams were used.[8] The JAMR authors reported

[8] https://github.com/jflanigan/jamr/tree/Generator

1. mkVP : VV → VP → VP

   P (*frame*$_A$ (:ARG1 (*var frame*$_B$)))
   R (want-01$_A$ (mkVP go-02$_B$))

2. mkCl : VP → Cl

   P (*var frame*$_A$)
   R (mkCl (mkVP want-01$_A$))

3. mkCl : NP → VP → Cl

   P (mkCl (mkVP (*frame*$_A$:ARG0)))
   R (mkCl :ARG0 (mkVP want-01$_A$))

4. mkNP : Quant → CN → NP

   P (:ARG0 (*var concept*$_A$))
   R (mkNP a_Quant (mkCN (b boy$_A$)))

5. mkCN : N → CN

   P (mkCN (*var concept*$_A$))
   R (mkCN boy_N$_A$)

6. Recursively merge :op1 .. :op$_n$ under :op1

   P (name (:op1 *literal*$_A$) (:op$_i$ *literal*$_B$))
   R (name (:op1 “New$_A$ York$_A$ City$_B$”))

7. mkPN : Str → PN
   mkNP : PN → NP

   P (*var* (name (:op1 *literal*$_A$)))
   R (mkNP (mkPN “New York City”$_A$))

8. Excise a node chain: var - NE type - :name

   P (*var* (*type* (:name mkNP)))
   R (mkNP)

9. mkVP : VP → Adv → VP
   mkAdv : Prep → NP → Adv

   P (mkVP (*frame*$_A$ (ARGn-*role*$_B$ mkNP)))
   R (mkVP (mkVP go-02$_A$)
   (mkAdv GOL_Prep$_B$ mkNP))

10. Ignore (delete) :wiki nodes.

11. Relabel frames, depending on their syntactic valence in the resulting AST. Thus, want-01 becomes want_VV (a verb-phrase-complement verb), in contrast to want_V2 (see Section 3.1), and go-02 becomes just go_V (an intransitive verb).

Figure 3: A sample set of AMR-to-AST transformation rules for converting the AMR in Figure 1 into the AST in Figure 2. P – pattern, R – result. The terms in italic (P) refer to regular expressions.

their original results on Gigaword corpus 5-grams (Gigaword licence required) which is known to improve the BLEU score by approx. 1 point.

3.4 First results

By applying the transformation ruleset (see Section 3.2), we were able to fully convert and linearize 12.3% of the 1293 evaluation AMRs. Additionally, we acquired partially transformed trees and, consequently, partially generated sentences for another 36% of the evaluation AMRs, but we did not include those sentences in the final submission, since large subtrees often got pruned. Instead, we replaced them by sentences acquired by JAMR Generator (see Section 3.3).

With the combined JAMR and GF-based system, we achieved Trueskill 107.2 and BLEU 18.82 in the preliminary official scoring. The key open question is if the 12.3% GF-generated sentences scored better or worse in comparison to the JAMR output by human-evaluated Trueskill.

By BLEU-scoring the GF-generated sentences apart from the JAMR-generated sentences, the BLEU scores are 11.35 and 19.18 respectively. This suggests that BLEU favors the corpus-driven JAMR approach which tries to reproduce the original input sentence, while the grammar-driven GF approach sticks to AMR more literally, producing a paraphrase of the input sentence. Therefore we are primarily interested in Trueskill and less concerned about BLEU.

4 Conclusion

The SemEval subtask on AMR-to-text generation has given us confidence that it is worth to develop further the GF-based approach. We are particularly pleased to see that AMR equipped with GF is emerging as a powerful interlingua for semantic cross-lingual applications. Although it is difficult to reach a large coverage in a short term, a grammar-based approach complemented with statistics from AMR and PropBank-annotated corpora is competitive with other approaches in a longer term, at least for use cases where adequacy is more important than fluency.

Acknowledgments

This work was supported in part by the Latvian State research programmes SOPHIS and NexIT, the H2020 project SUMMA (grant No. 688139), and the European Regional Development Fund grant No. 1.1.1.1/16/A/219. We also thank Aarne Ranta, Alexandre Rademaker and Alastair Butler for encouraging and helpful discussions.

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for Sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*.

Guntis Barzdins and Didzis Gosko. 2016. RIGA at SemEval-2016 Task 8: Impact of Smatch extensions and character-level neural translation on AMR parsing accuracy. In *Proceedings of the 10th International Workshop on Semantic Evaluation*.

Alastair Butler. 2016. Deterministic natural language generation from meaning representations for machine translation. In *Proceedings of the 2nd Workshop on Semantics-Driven Machine Translation*.

Jeffrey Flanigan, Chris Dyer, Noah A. Smith, and Jaime Carbonell. 2016. Generation from Abstract Meaning Representation using Tree Transducers. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Normunds Gruzitis and Guntis Barzdins. 2016. The role of CNL and AMR in scalable abstractive summarization for multilingual media monitoring. In *Controlled Natural Language*, Springer, volume 9767 of *LNCS*.

Roger Levy and Galen Andrew. 2006. Tregex and Tsurgeon: Tools for querying and manipulating tree data structures. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*.

Jonathan May and Jay Priyadarshi. 2017. Semeval-2017 task 9: Abstract meaning representation parsing and generation. In *Proceedings of the 11th International Workshop on Semantic Evaluation*.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics* 31(1).

Aarne Ranta. 2004. Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming* 14(2).

Aarne Ranta. 2009. The GF Resource Grammar Library. *Linguistic Issues in Language Technology* 2(2).

Aarne Ranta. 2011. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford.