EasyChair Preprint

№ 6414

Semantic Parsing of Geometry Statements Using Supervised Machine Learning on Synthetic Data

Salwa Tabet Gonzalez, Stéphane Graham-Lengrand,
Julien Narboux and Natarajan Shankar

August 26, 2021

Semantic parsing of geometry statements using supervised machine learning on synthetic data

Salwa Tabet Gonzalez[1], Stéphane Graham-Lengrand[2],
Julien Narboux[3], Natarajan Shankar[2]

[1] École Polytechnique, France
[2] SRI International, USA
[3] ICube, UMR 7357 CNRS, University of Strasbourg, France

Introduction

Computerization of mathematical text is a long-term challenge for the Mathematical Knowledge Management (MKM) community. It has wide-ranging applications: semantic search into mathematical documents, proof checking, combining proofs and computations, checking computations, automated theorem proving, ...

In this extended abstract, we report on our ongoing work on the automated translation of high-school geometry statements into a formal language of syntax trees.

Such an automated translation could be used to help the user of an interactive or automated theorem prover remember the syntax or definitions used in a formal setting, similarly to the tools that translate a drawing of a symbol to the LaTeX description[1]. Such a translator could also be used in tutoring environments where the use of a formal language, or even a *controlled* natural language, is prohibitive. If the approach is successful, then it could be extended to the formalization of *proofs* for applications in education for example.

1 Overall description of the approach

Our approach is based on Arsenal [Ars19], a framework developed at SRI International for building domain-specific translators from natural language to structured representations, namely syntax trees. Each domain specific translator is obtained by supervised machine learning. This raises the issue of the labeled dataset used for training, for applications where ground truth data may be sparse. Arsenal's approach to this issue is to generate synthetic datasets from the very specification of what Arsenal's output trees should look like and from a pretty-printer of such trees into natural language. The former is given in the form of a domain-specific *grammar* specifying which syntax trees are acceptable as output, and technically expressed as a collection of OCaml algebraic datatypes (ADTs). The generation of synthetic datasets uses a customized version[2] of the OCaml meta-programming library `ppx_random` for producing random inhabitants of ADTs. Arsenal also integrates type-checking at runtime, i.e. in the production, by the trained model, of syntax trees, which are thus well-typed by construction.

More information on Arsenal can be found in its documentation [EGLSY20], two figures of which are reproduced in Appendix A with the description of Arsenal's core mechanisms. The grammar and its pretty-printer are the cornerstone of every new Arsenal application, and so far, Arsenal has been applied to several domains such as systems requirements (2018-2019) and 5G standards (2020-present).

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

[1] `https://detexify.kirelabs.org/classify.html`
[2] `https://github.com/disteph/ppx_deriving_random`

For this new application to geometry:

- We designed a grammar for geometry statements in first-order logic, more specifically in the form of *coherent implications* (see, e.g., [DN15]), taking as inspiration the concepts used in French high-school math textbooks and those present in the GeoCoq library[3]. See Section 3 for details.
- We equipped the grammar with probability biases that Arsenal's random tree generator uses to sample the tree space.
- We programmed a pretty-printer to translate trees from the grammar into natural language (in this case, French). See Section 4 for details.
- We used Arsenal's labeled dataset generator, leveraging the above, to produce a synthetic training set of 10^6 statements. Examples of such generated statements, in both natural language and syntax trees (S-expressions), are given in Appendix B.
- We implemented an *entity processor* for geometry, to reduce the difficulty of the learning task (see Section 2).

Arsenal provides a domain-generic infrastructure to train a seq2seq model on the training data (see Appendix **??**), as well as providing a GUI and a bash script to experiment with the trained model and the entity processor. Such experimentations in the case of geometry are on-going.

- In addition, we manually built a corpus of about 80 statements presented in several form of natural language (with or without *anaphoras*) as well as in first-order logic. Examples of corpus statements are given in Appendix C.

2 Entities

While the model's objective is to understand how the complex arrangement of words in a sentence describes predicates and functions in the grammar, some individual words or lexemes can directly be understood, in pre-processing, as *entities* that correspond to terminals in the grammar. For instance, in order for Arsenal to correctly translate the following two sentences as syntax trees:

"Si O est le milieu de [AB] alors O appartient à [AB]."

"Si P est le milieu de [CD] alors P appartient à [CD]."

it suffices to have a model that can correctly translate

"Si Point0 est le milieu de [Point1 Point2] alors Point3 est le milieu à [Point4Point5]."

where Point0, Point1, Point2, Point3, Point4, Point5 are *placeholders* abstracting the *entities* O, A, B, O, A, B in the first sentence, and P, C, D, P, C, D in the second.

Therefore, Arsenal reduces the size of the space of sentences and syntax trees that the model operates on, by pre-processing its runtime input with an *entity processor* that detects entities and replaces each of their occurrences by a typed and indexed placeholder. Indeed, while the example above only shows entities of one type, namely *Point*, Arsenal's entity placeholders are typed and, within each type, they are indexed incrementally from left to right in the resulting sentence. Beside variable names, another type of entities in geometry (and in mathematics in general) characterizes constant numbers occurring in input sentences, as in "La distance est supérieure à 10."

In the entity processor that we developed for geometry, the typing is done by recognizing constant numbers and tagged names first, such as lines ((d_1) or (AB)), rays ($[AB)$), segments ($[AB]$) and angles ($\angle ABC$). Then, single-entity names are replaced: for example, circle names are recognized if the name is preceded by *"the circle"* (in French, *"le cercle"*). If the entity processor fails to type a specific name, it is by default considered as a point.

Using entity placeholder in Arsenal's runtime means that the labeled dataset used to train the model only features placeholders rather than actual entities. This in turn means that the generator of syntax trees has a much smaller space to sample from.

3 An algebraic datatype for geometry statements

An Arsenal grammar specifies the form of syntax trees modelling concepts, and relations between concepts, that together describe an application domain. After the entity types are declared, the grammar file describes a context-free grammar by declaring a number of algebraic datatypes that constitute the grammar's syntactic categories.

[3] `https://geocoq.github.io/GeoCoq/`

In our case, a syntax tree is of type `formula`, encoding coherent implications of the form

$$\forall \vec{x} \left(\bigwedge_i H_i \Rightarrow \exists \vec{y} \left(\bigvee_j \bigwedge_k A_{j,k} \right) \right).$$

The formula constructor `Fml` correspondingly takes 4 arguments:

- A list of universally quantified objects $\vec{x}$;
- A list of atoms $H_1, \ldots, H_n$;
- A list of existentially quantified objects $\vec{y}$;
- A list of lists of atoms $(A_{1,1}, \ldots, A_{1,p_1}), \ldots, (A_{m,1}, \ldots, A_{m,p_m})$.

Each of these components has a type: bindings are collections of objects, whereas hypotheses and conclusions are conjunctions of atoms, applying predicates to objects. These predicates describe relations between different typed objects, which can be points, lines, rays, segments, planes, angles, triangles, quadrilaterals, polygons, circles or numbers (especially distances).

Rather than designing a minimalistic grammar with only few basic constructions from which others can be defined, we intentionally defined a rich grammar reflecting the variety of concepts that we expect to find in natural language. Thus shortening the distance between the formal and informal languages simplifies the development of the pretty-printer and the difficulty of the learning task.

The generation of a synthetic dataset by Arsenal starts with the production of random inhabitants, i.e. syntax trees, of type `formula`, by the OCaml library `ppx_random`. The Arsenal component leveraging the library is domain-generic, i.e. it does not need to be changed for different application domains, since it takes a domain-specific grammar as input and produces a syntax tree generator as output (this is described in the upper part of Fig.2, in Appendix B).

At every stage of the depth-first procedure, the generator builds a sub-tree by first selecting its root, i.e. a constructor of the relevant type, according to a probability distribution annotating the type declaration.

Indeed, in order to make sure that the generated data does cover the typical geometry statements that will be used in practice, it is convenient to bias the generation so that some constructs are more likely to be generated than others. We provide these biases for high-school geometry. For example, we specify that segments are more likely to be referenced by the names of the endpoints rather than the name of the segment itself. Also, a normal user will more often manipulate definite points than the circumcenter of a triangle.

Another aspect guiding the design of probabilities is the sub-expresion's depth: in natural language we rarely build sentences with great depths and tend to flatten trees by splitting them into several sentences. In our customized version of `ppx_random`, the probability of constructors to be picked can vary with depth, and indeed with greater depth we increase the probabilities of non-recursive constructors, typically constructors without arguments.

4 The pretty printer

The second task to do when applying Arsenal to a new domain is to pair each grammar construct with several natural language phrasings for it. These illustrate "typical ways" in which the grammar construct could be described in natural language (which often offers numerous ways of expressing the same content). Arsenal's pretty-printer then non-deterministically compounds the NL phrasings, so as to produce a randomly generated NL sentence for each of the randomly generated trees. Those pairs constitute the labeled dataset with which the model can be trained.

We wrote such a randomized pretty-printer for our geometry grammar. Arsenal provides a library for minimizing the manual effort to write it. Once again, we determined biases for choosing the most likely phrasings of each geometric construction, and thanks to Arsenal, this indicates how the synthetic dataset is populated. Appendix B provides some examples of synthetic pairs.

4.1 Preprocessing the syntax tree

As stated before, universally and existentially quantified variables are named objects in our application. These named objects sometimes are further defined in the following predicates (hypotheses or conclusions), which would be translated in natural language by adjectives if unary. For example, the first-order logic formula

$$\forall ABC, Triangle(ABC) \wedge Equilateral(ABC, A) \Rightarrow AB = AC = BC$$

can be translated into the less natural sentence

$$\text{For all triangles } ABC, \text{ if } ABC \text{ is equilateral, then } AB = AC = BC$$

or the more processed sentence

$$\text{For all equilateral triangles ABC, AB = AC = BC.}$$

This is done by checking the predicates involving the named objects, adding information about their unary adjectivation and then removing the corresponding predicates. Also, another preprocessing we added to better emulate natural language is factorization: again the first order logic formula

$$\forall A, \forall B, \forall C, Midpoint(B, AC) \Rightarrow AB = BC$$

can be translated into the less natural sentence

$$\text{For all points } A, \text{ points } B, \text{ points } C, \text{ if B is the midpoint of } [AC] \text{ then } AB = BC$$

or the more processed sentence

$$\text{For all points } A, B, C, \text{ if B is the midpoint of } [AC] \text{ then } AB = BC.$$

This tranformation takes place after adding adjectives, in order to group together objects with same characteristics which are defined contiguously.

Other preprocessing tasks are handled here, but they are specific to the French language. For instance, the keywords *"for all"* and *"such that"* are not invariant in French: they can be declined in their feminine, plural and feminine-plural forms. To match the correct form, there is a need to inspect the object that is qualified by these keywords.

4.2 Printing the transformed tree

After having prepared the syntax tree to be printed, the actual formatting takes place. The purpose of the pretty-printing file is to provide a function that takes a syntax tree and turns it into an NL sentence. That file is dependent on the grammar file because it makes references to it: each type has a pretty-printing function that recursively calls other functions, and these functions rely on pattern-matching of the constructor to build the sentence.

One of the objectives of the syntax tree generator and pretty-printing is to produce an arbitrarily large set of labelled data. If for each syntax tree there were only one way to write it in natural language, the model would not be very robust. Since we want to capture the fact that there is often more than one way to say something (in particular, several ways to express the same syntax tree), we allow the randomisation of the pretty-printing functions. Hence, every time such a function is called on a given input syntax tree, it produces at random one of the many phrasings that describes it in natural language. This is done in a lazy manner: without it, the pretty-printing function would, first, produce all possible NL sentences for the (whole) tree, and then pick one of them according to the probabilities. That would be far too slow. Instead, our lazy pretty-printing for a tree first selects a phrasing before pretty-printing the sub-trees (if any). Only one sentence for the whole tree is produced by one pretty-printing call.

Again, we put bias in the generation of NL sentences. There are several ways to express the same syntax tree, but some of them are more common and sound less artificial, even if they all are gramatically correct. Therefore, for nearly all grammar constructors, we put probabilities on the various phrasings for it, trying to reflect the most natural usage. For example, when pretty-printing the construct $Line(A, B)$, which is a line defined by two points, we would naturally phrase it as *"(AB)"* rather than the very heavy *"the line delimited by the points A and B"*, even though both are correct. Both expressions exist in our pretty-printer, but the former benefits from a greater bias than the latter.

Other features of natural language are handled here. For example, to avoid printing non-essential parentheses we deal with operation priority: even if the sentence $(5 \times ((4+7)-3))$ is technically correct, in a natural language one would write $5 \times (4 + 7 - 3)$.

Related work and conclusion

Our approach shares some similarities with the one presented in [WBKU20], which also leverages a translation from formal expressions to natural language to build a training set. But there are also differences: the Arsenal toolkit randomly generates syntax trees to create arbitrarily large training sets based on user-defined probabilities, and its pretty-printers translating to natural language are also non-deterministic, being able to generate many natural language variants of the same grammar constructs; in the present endeavor, we also focus on the domain of high-school geometry.

Automated solving of mathematical problems has been a challenge for natural language processing community [MGN07]. For geometry, they are different lines of works based on extraction of geometric predicates either from the text or also from figures [GYW19, SHFE14, SHF$^+$15]. Our geometry grammar could also be compared to GeometryNet [MGN07].

Controlled natural language have also been proposed [CFK$^+$10, CM13]. The drawbacks include a decrease of legibility for humans, and requiring the same discipline from users as if they were reading and writing code. Arsenal aims at retaining a more flexible use of natural language to express formal content, so that this content can be manipulated by users who may not be coders, while being amenable to automated processing for, e.g., formal modeling and analysis.

In this preliminary work, we have contributed a collection of algebraic datatypes to capture the usual statements of high-school geometry along with a natural language pretty printer for this specific domain. For evaluation purposes, we have created a corpus of pairs of natural language statements (currently in French) and their formalization. This is ongoing work and, although model training and experimentation are still to be conducted, experience on prior domains of application such as systems requirements and 5G standards give us confidence that the approach is worth exploring.

References

[Ars19] The Arsenal software, 2019. `https://github.com/SRI-CSL/arsenal-base`

[CFK$^+$10] M. Cramer, B. Fisseni, P. Koepke, D. Kühlwein, B. Schröder, and J. Veldman. The Naproche Project Controlled Natural Language Proof Checking of Mathematical Texts. In N. E. Fuchs, editor, *Controlled Natural Language*, pages 170–186. Springer Berlin Heidelberg, 2010. bibtex: 10.1007/978-3-642-14418-9_11.

[CM13] N. C. Carter and K. G. Monks. Lurch: a word processor that can grade students' proofs. In *Workshops and Work in Progress at CICM*, 2013

[DN15] R. Dyckhoff and S. Negri. Geometrisation of first-order logic. *Bulletin of Symbolic Logic*, 21(2):123–163, 2015.

[EGLSY20] D. Elenius, S. Graham-Lengrand, N. Shankar, and E. Yeh. Arsenal, 2020. `https://github.com/SRI-CSL/arsenal-base/blob/master/doc/Arsenal-principles.pdf`

[GYW19] W. Gan, X. Yu, and M. Wang. Automatic Understanding and Formalization of Plane Geometry Proving Problems in Natural Language: A Supervised Approach. *International Journal on Artificial Intelligence Tools*, 28(04):1940003, 2019.

[MGN07] A. Mukherjee, U. Garain, and M. Nasipuri. On Construction of a GeometryNet. In *Proceedings of the 25th Conference on Proceedings of the 25th IASTED International Multi-Conference: Artificial Intelligence and Applications*, AIAP'07, pages 530–536. ACTA Press, 2007. event-place: Innsbruck, Austria.

[SHF$^+$15] M. Seo, H. Hajishirzi, A. Farhadi, O. Etzioni, and C. Malcolm. Solving Geometry Problems: Combining Text and Diagram Interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1466–1476. Association for Computational Linguistics, 2015.

[SHFE14] M. J. Seo, H. Hajishirzi, A. Farhadi, and O. Etzioni. Diagram Understanding in Geometry Questions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), 2014. Section: AAAI Technical Track: Vision

[WBKU20] Q. Wang, C. Brown, C. Kaliszyk, and J. Urban. Exploration of neural machine translation in autoformalization of mathematics in Mizar. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 85–98. ACM, 2020.

A Arsenal's core mechanisms

Arsenal's run-time component takes as input a sentence, which is stripped of its entities, as described in Section 2. Then, the model builds a description of the corresponding syntax tree, and the entities are again substituted in the tree to finally obtain a tree with entities, as shown in Fig.1.

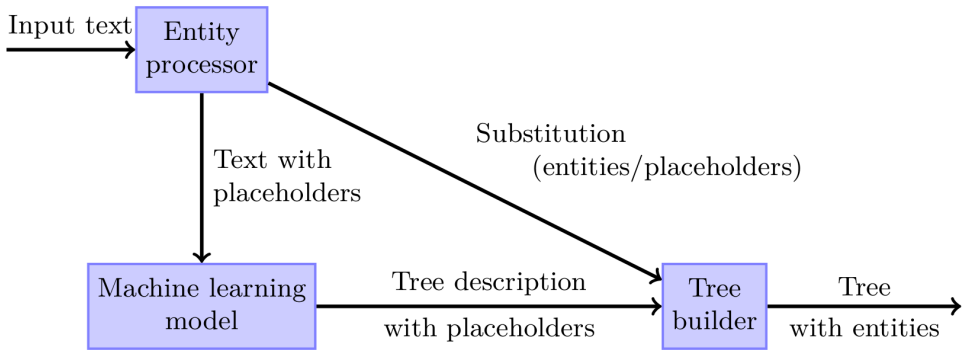

Figure 1: Arsenal's run-time component

Training the model from the labeled dataset is performed using the standard techniques of supervised machine learning. Arsenal currently offers a training and runtime infrastructure for a variant of sequence-to-sequence (seq2seq) models equipped with an attention mechanism, which have been used successfully for machine translation from natural language to natural language.

Such models are made of an encoder and a decoder. The encoder is a Recurrent Neural Net (RNN) that keeps an internal state (a vector of reals); as the input sentence is recursively traversed from one end to the other, the internal state is updated with every word that is read (and encoded as a vector of reals). The history of the encoder's internal state is recorded. The decoder produces a stream of tokens: in our case, the tokens make up the Polish notation for the output tree. It also has an internal state, and an attention mechanism that focuses on certain parts of the encoder's internal state history. The decoder produces one token at a time, choosing it according to the last token it produced, its internal state and the attention, both of which are updated before producing the next token. In Arsenal, the sequence-to-sequence approach has been adapted to produce tree descriptions in Polish notations, dynamically forcing the output tokens to describe a tree that is well-formed with respect to Arsenal's output grammar, and changing the halting condition of the output stream to make is stop as soon as the produced stream describes a complete, well-formed tree.

When using the training and runtime infrastructure for seq2seq, the user can fix the parameters such as the number of layers, the number of nodes per layer, etc.

Fig. 2 describes the Arsenal pipeline for producing a training dataset and training the model with it.

B Example of Synthetic Data

Below are 3 examples of synthetic statements that are generated by Arsenal from the geometry grammar. Each example shows, first, the natural language sentence, and, second, the randomly generated tree from which it comes, as an S-expression that starts with the constructor for coherent implications `Fml` (see Section 3). In the three examples, the lists of quantifiers are empty.

```
Si (_Line_string_000) et (_Line_string_001) sont coplanaires, alors les points
_Point_string_000, _Point_string_001 et _Point_string_002 sont égaux.

(Fml Nil
```

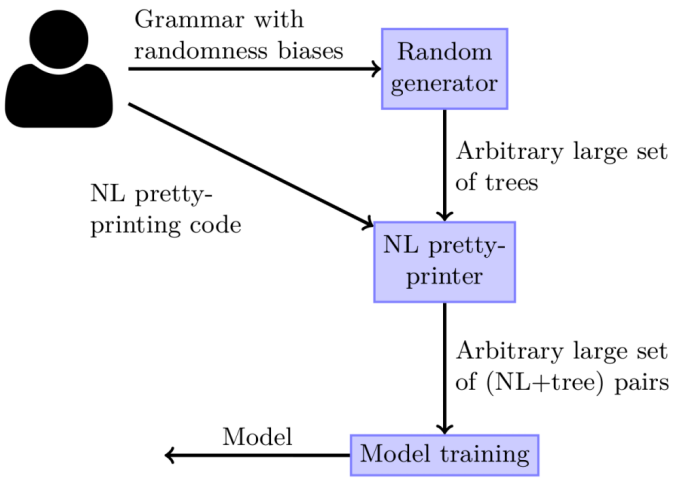

Figure 2: Synthetic dataset generation and model training

```
 (List
  (Coplanar_lines
   (List (Named_line Line_string_000) (Named_line Line_string_001))))
 Nil
 (List
  (List
   (Same_point
    (List (Named_point Point_string_000) (Named_point Point_string_001)
     (Named_point Point_string_002))))))
```

```
Si _Point_string_000 ∈ (_Point_string_001_Point_string_002), alors les droites
(_Point_string_003_Point_string_004) et (_Line_string_000) sont sécantes.

(Fml Nil
 (List
  (Incid_line (Named_point Point_string_000)
   (Line (Named_point Point_string_001) (Named_point Point_string_002))))
 Nil
 (List
  (List
   (Is_intersecting_line_line
    (Line (Named_point Point_string_003) (Named_point Point_string_004))
    (Named_line Line_string_000)))))
```

```
Si _Point_string_000 = _Point_string_001, alors la médiatrice de
[_Point_string_002_Point_string_003] est (_Point_string_004_Point_string_005).

(Fml Nil
 (List
  (Same_point
   (List (Named_point Point_string_000) (Named_point Point_string_001))))
 Nil
 (List
  (List
```

```
(Same_line
 (List
  (Perp_bisector
   (Segment (Named_point Point_string_002) (Named_point Point_string_003)))
  (Line (Named_point Point_string_004) (Named_point Point_string_005)))))))
```

C Examples taken from our corpus

Si O appartient à [AB] et OA = OB alors O est le milieu de [AB].
Si A et A' sont symétriques par rapport au point O alors le point O est le milieu de [AA'].
Si (d) est la médiatrice du segment [AB] alors (d) coupe le segment [AB] en son milieu.
Si ABC est un triangle rectangle d'hypoténuse [AB] alors le centre de son cercle circonscrit est le milieu de [AB].
Si ABCD est un losange alors (AC) $\perp$ (BD).
Si (d) est la médiatrice du segment [AB] alors (d) est perpendiculaire à [AB].
Si C appartient au cercle de diamètre [AB] alors ABC est rectangle en C.
Si ABCD est un parallélogramme et (AC) $\perp$ (BD) alors ABCD est un losange.
Si ABCD est un parallélogramme et AB = BC alors ABCD est un losange.