Towards an Annotation Standard for STEM Documents

Datasets, Benchmarks, and Spotters

Jan Frederik Schaefer[0000−0003−2545−4626] and Michael Kohlhase[0000−0002−9859−6337]

Friedrich-Alexander-Universität Erlangen-Nürnberg

Abstract. When publishing papers, researchers in mathematics and related disciplines typically focus on the presentation, i.e. type-setting, of their ideas and provide little semantic information. This impedes the development of services that benefit from semantic information, such as semantic search and screen readers for vision-impaired researchers. As a remedy, there have been attempts to infer semantic data from already published papers using small programs that we call *spotters*. Unfortunately, there is no standardized format for semantic annotations and spotter authors typically invent their own format. This leads to two problems: *i*) there is no ecosystem of tools for common tasks like the visualization of results or the manual annotation of a gold standard, and *ii*) re-using, evaluating and combining results becomes very difficult.
In this paper, we address these issues by describing a standardized, flexible way to represent semantic annotations, using semantic web technologies and, in particular, the Web Annotation standard. Furthermore, we describe SpotterBase, a set of tools to help with processing the annotations and creating new ones.

1 Introduction

With the number of publications in STEM (Science, Technology, Engineering and Mathematics) rising rapidly, the challenge of managing and efficiently accessing the knowledge they carry becomes ever more relevant. A variety of services could help, for example

- *Specialized formula search engines* can help discover formulae, which traditional search engines are notoriously bad at. For example, we might be looking for a closed-form expression of $\sum_{n=0}^{\infty} n\frac{2^n}{n!}$. A unification-based search engine could unify that expression with the left-hand-side of the equation $\sum_{k=0}^{\infty} k\frac{z^k}{k!} = ze^z$.
- *Active documents* provide functionality for interacting with a document. They could, for example, convert units on demand, show where a variable was declared, or allow inserting concrete values into formulae for computation.
- *Screen readers* can read out documents for vision-impaired researchers.

– *A semantic document checker* can help authors find certain types of errors, e.g. by pointing out that in *"a density of 12 g/m"* the unit does not match the expected SI dimension of mass per volume.

Such services are easy to realize if the semantic information implied in the text is made explicit: a formula search engine delivers better results if it knows what identifiers stand for and how they are quantified, an active document can only convert units if it knows what the units are, screen readers can read formulae better if they understand them (e.g. *"$|x|$"* could be *"the magnitude of x"*), etc. In practice, such information is rarely available because publications are typically type-set for human consumption with little regard for explicit semantic annotations, and if such information is present in the source, e.g. by using LaTeX packages like siunitx.sty, it gets lost in the compilation to PDF.

A well-known remedy is to add semantic annotations to existing publications with **spotters**: programs that search a corpus for occurrences of a particular semantic phenomenon. A diverse collection of spotters can build up a large set of semantic annotations, which can then be used to create or improve semantic services. Over the years, a number of spotters has been implemented, but a shared collection of semantic annotations that would allow to synergize remains elusive. The main reason seems to be the lack of an agreed-upon standard for annotations, which has led spotter authors to either share their annotations in a custom format incompatible with other annotation sets, or to abandon the idea of publishing a re-usable dataset altogether.

The lack of a standard leads to another problem: based on our experience of supervising bachelor's and master's theses, a large part of their effort goes into building tools for visualizing results, manually annotating a test dataset, automatically evaluating their results against the test dataset, etc. A shared standard for annotations would allow for the development of re-usable tools for such tasks.

The natural language processing (NLP) community benefits from a long tradition of annotation tasks and benchmarks, but very few of them exist for mathematical language specifically. A note-worthy exception are the math information retrieval tasks at NTCIR [AK20] and more recently at CLEF [Man+22]. An annotation standard could facilitate the development of tasks and benchmarks for processing STEM documents. Effectively, anything that we might want to have a spotter for could be turned into a task. We hope that an ecosystem of datasets, tools and benchmarks facilitated by a common representation will incentivize others to adopt it.

Contribution We present a flexible representation for semantic annotations based on the Web Annotation Standard [Webb]. We have successfully used it for different types of annotations on the arXMLiv corpus [Gin20]. Furthermore, we present SpotterBase[1], a collection of tools to create and work with annotations in that format.

[1] Open source; code at `https://github.com/jfschaefer/spotterbase`

Overview In Section 2, we discuss related work. Afterwards, we discuss our approach of accumulating semantic information with spotters in Section 3, followed by a discussion of what documents we are interested in (Section 4). In Section 5 we will discuss the annotation format and in Section 6 SpotterBase, a set of tools to create and work with such annotations. Section 7 discusses our experience with this setup so far and Section 8 concludes the paper.

2 Related work

Text annotations for natural language processing When processing texts without formulae, there is a more-or-less generally accepted plaintext representation. This simplifies the processing and allows for either stand-off annotations via string offsets or in-document annotations via simple markup. Furthermore, such texts can be conveniently represented as a sequence of tokens (words), which can then be annotated with tags. The family of CoNLL formats, which is often used for NLP tasks (see e.g. [Con]), is based on this idea. Formulae complicate this as there is no generally accepted way to represent complex formulae in plain text or as a token sequence, and replacing entire formulae with a single token loses relevant information.

Manual annotation tools Even though our goal is to create annotations automatically, it is also necessary to manually create annotations for evaluation and training of machine learning models. We can distinguish between general-purpose annotation tools and more specialized ones. General-purpose tools would, e.g., be PDF viewers that support annotating documents or the hypothes.is tool [HYP], which allows users to annotate web pages. More specialized tools were developed for NLP tasks, such as part-of-speech tagging or named entity recognition. Examples of this are WebAnno [Cas+16] (not to be confused with the Web Annotation recommendations) and brat [BR]. There are also tools that are specialized in the annotation of mathematical language and can handle formulae: KAT [Gin+15] uses semantic web technologies for annotating HTML documents and can be customized for different annotation tasks. MioGatto [Asa+21] is a much more recent system, specifically designed for annotating the grounding of identifiers. KAT and MioGatto each have their own, custom format for representing and storing annotations.

Semantic authoring Our work is based on the assumption that authors do not put effort into providing semantic annotations. However, there are attempts to enable authors to supply semantic markup, e.g. by using the sTeX package [CICM22], which can be used to annotate LaTeX sources via semantic macros.

In-document annotations Annotations can be stored either in-document or in a separate database that references the documents, which is the approach we follow (Section 3). RDFa [Her+13] is a common approach to store metadata in HTML documents. When it comes to formulae, another option is to use some of the

features provided by MathML, the Mathematical Markup Language. MathML is used to represent formulae in HTML5 and it provides Content MathML to annotate formulae with a semantic representation. There is ongoing work to extend MathML with "intents" as a way to describe the intended meaning of a (sub-) formula to improve accessibility (e.g. to help screen readers read out formulae properly). While we are primarily interested in stand-off annotations, adding intent attributes to the documents from the semantic annotations would be an interesting application once MathML intents have stabilized.

Full formalization Our goal is to accumulate annotations of different semantic phenomena. A much more ambitious goal is the full formalization of publications, i.e. translating them into a logic. A well-known example of this is the proof of the Kepler conjecture, which has been fully formalized to the extent that its correctness could be automatically verified [Hal+17]. A less ambitious variant is to only formalize the key results of the paper, as e.g. envisioned by the formal abstracts project [FA]. There have also been attempts to use deep learning approaches for formalizing mathematics (e.g. [Wan+20]), but the automated formalizing of STEM publications with reasonable accuracy appears to be beyond the state of the art. While it is conceivable that semantic annotations could aid automated formalization endeavours in the future, this is not our goal in this paper. Indeed, we consider the semantic annotation approach largely independent of full formalization efforts. We observe that a full formalization does not immediately satisfy all information needs for semantic services, unless it is tightly linked to the original document for human consumption.

3 Accumulating Semantic Information

Our approach is centered around a shared collection of semantic annotations. Figure 1 illustrates the setup. Using stand-off annotations — as opposed to in-document annotations — has the advantage that there can be many independent contributors of annotations.

When a corpus is imported, there is typically a lot of metadata resulting in document-level annotations. Examples are: titles, authors, classifications, publication years, etc. The main contributor of annotations, though, are **spotters**, of which we distinguish three different types:

- *Simple spotters* process documents and create annotations without using the results of other spotters. An example of this would be a spotter that detects references to mathematical objects in the text (e.g. that the string *"Abelian group"* in a document refers to Abelian groups). Another example would be a part-of-speech tagger.
- *Hybrid spotters* additionally consider annotations made by other spotters. For example, a spotter for identifier declarations (*"let G be an Abelian group"*) might benefit from the annotation that *"Abelian group"* refers to a mathematical object.

– *Meta spotters* only act on annotations. Such spotters can combine the results of different spotters into a new set of annotations, resolving conflicts in the process. Let us take the example of a text containing the identifier *"C"*. A spotter for units might annotate it as referring to the unit Coulomb, while a different spotter might link it to a previously declared variable C. A meta spotter could resolve this, e.g., by always prioritizing the interpretation that it references a declared variable. In this case it could also be reasonable to take further annotations into consideration, such as the topic of the publication.

Hybrid and meta spotters tend to find more complex semantic information than simple spotters. In the past, the lack of an agreed-upon annotation format made it difficult to create anything other than simple spotters and prevented the build-up of a dataset of diverse semantic annotations that could be harvested for semantic services.

While spotters provide the bulk of annotations, we also need a small amount of human-created annotations — to evaluate spotters and, if machine learning is used, as training sets. As we can have different sources of the same kind of annotation (a human, a spotter, a different spotter, etc.), we also need to track where the annotations come from.

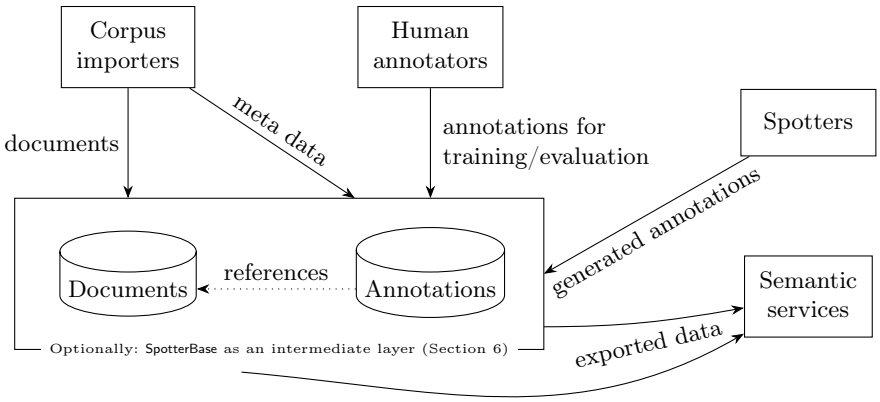

Fig. 1. Overview of the setup.

4 The Document Corpus

The setup sketched in Figure 1 does not impose any restrictions on the types of documents that are annotated and could, in principle, work for documents in any format. Our focus, however, is on STEM publications in the HTML format. HTML might seem like an unintuitive choice — after all, STEM publications are mostly written in LaTeX or Office and distributed as PDF documents. However, neither format is particularly well suited for semantics extraction. Depending

on the author, macro expansion (or rather a full TEX engine) is required to reasonably process a LATEX document. PDF and Office documents, on the other hand, are too focused on presentation, i.e. placing symbols on a page, which makes text processing very difficult, especially, if formulae are involved. Another advantage of HTML over LATEX and PDF is that it can be used much more easily for the development of semantic services.

We have mostly worked with the **arXMLiv** corpus [Gin20], but none of the presented ideas are specific to it. ArXMLiv has been created by processing the LATEX sources of arxiv.org with the LaTeXML [Mil] tool to obtain HTML5 documents with MathML for the formulae. LaTeXML tries to carry any semantic information from the TEX sources into the resulting HTML5 document, e.g. if "semantic" macros like \sin are used. With about 1.6 million documents in the last release, the corpus is large enough that scalability becomes a major concern.

5 Annotations as RDF Triples

The Resource Description Framework, short RDF, uses triples of URIs to represent data [RDF]. A triple (s, p, o) can be thought of as an edge from s to o with label p. A collection of triples therefore encodes a directed graph. Triples can be stored in specialized triple stores and queried with SPARQL queries. We use this well-established framework to encode, store and retrieve annotations. In particular, our encoding is based on the recommendations by the W3C Web Annotation Working Group [Webb].

To illustrate the encoding of an annotation, let us assume that a document doc00.html contains the text *"it has a density of 1292.1 gm^{-3}"* and we want to annotate that *"1292.1 gm^{-3}"* is a quantity consisting of the scalar 1292.1 and the unit grams per cubic meter. In the Web Annotation recommendations, the main components of an annotation are the **target**, which describes what is annotated (in this case a particular text fragment), and the **body**, which contains some information about the target (in this case what the quantity is). Additionally, the annotation may be associated with metadata indicating, for example, who has created the annotation. Figure 2 sketches the RDF graph for the example annotation. We use the oa: prefix for the Web Annotation vocabulary [Weba], which is based on the Open Annotation vocabulary. The prefix sb: (SpotterBase) is used for URIs from our extension.

In the following subsections, we will take a closer look at how the target and the body of the annotation are represented and how a triple store of such annotations can be queried using SPARQL (Section 5.4).

5.1 Annotation targets

The annotation target describes what we want to annotate. That could be an entire document, e.g. if we want to annotate its language, but often we only want to annotate part of a document, like in the example shown in Figure 2. The Web

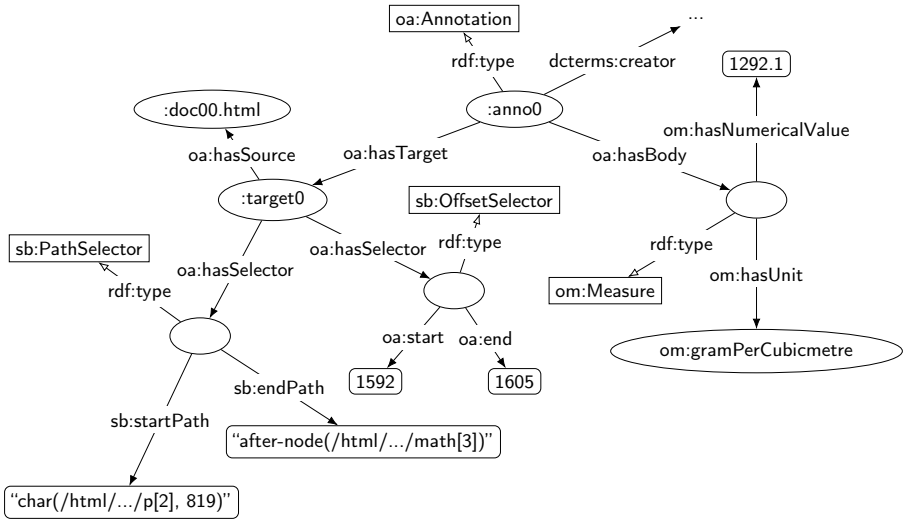

Fig. 2. An example annotation.

Annotation specification provides **selectors** to specify what part of the document we are interested in. However, the provided selectors did not work well in our setting as we need to select both text offsets and XML nodes with high precision. Instead, we specify three new selectors: sb:PathSelector, sb:OffsetSelector and sb:ListSelector. It is imaginable that in the future yet another type of selector should be supported, for example, to annotate figure contents.

The **sb:PathSelector** selects a continuous document fragment. The start and end of the selection are specified as strings:

1. "char(p, n)" points to the n-th character in the node referenced by the XPath p (an XPath [XPa10] is a standardized way to select an XML node in a document).
2. "node(p)" points to the node that is referenced by the XPath p.
3. "after-node(p)" points to whatever comes right after the node that is referenced by the XPath p.

after-node was introduced because the end is not included in the selected fragment (to be compatible with similar selectors in the Web Annotation recommendations). Note that it is necessary to be able to select HTML nodes, not just text characters. For example, the formula "$\sqrt{x}$" has the MathML representation <msqrt><mi>x</mi></msqrt> and it makes a big difference whether the <msqrt> or the <mi> ("math identifier") node are selected.

The **sb:OffsetSelector** can select fragments with the same granularity as the sb:PathSelector, but it specifies the beginning and end of the fragment with integer offsets. Providing multiple, equivalent selectors improves the chances that a consumer can process them. The sb:PathSelector is designed to be easy to use by a wide range of tools, while the sb:OffsetSelector makes it possible to compare the order of annotations easily (and without the need to load the document).

For example, we can use it in SPARQL queries to check if an annotated range lies inside another annotated range (see also Section 5.4).

Before discussing the sb:ListSelector, which can select discontinuous fragments, we will take a look at the annotation body.

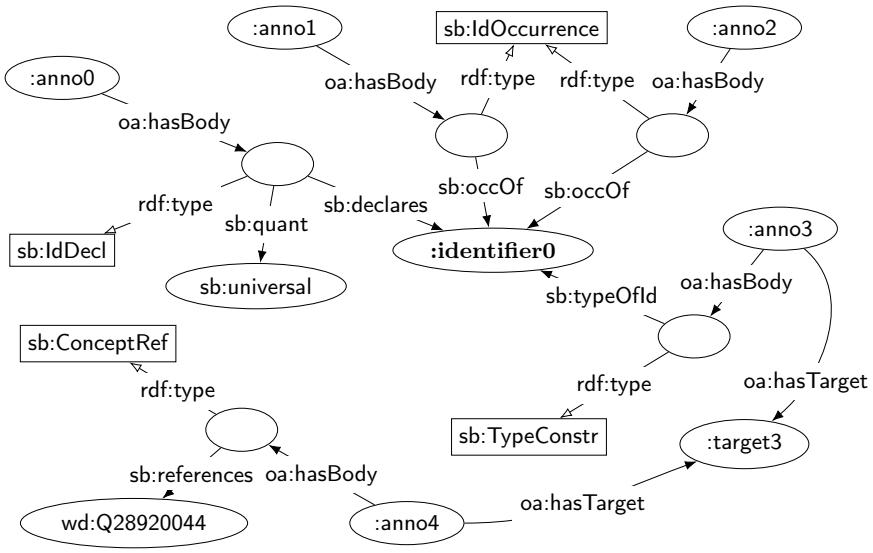

Fig. 3. Multiple annotations referencing the same identifier. For brevity, most targets are omitted and the sb:∗ URIs are simplified.

5.2 Annotation bodies

The body of an annotation describes what information is attached to the annotation target. For some annotation tasks, we can develop a standardized representation of the body. For example, many annotation tasks simply require a tag as a body, such as the theorem tag to annotate a paragraph as a theorem. The advantage of annotations with a standardized body is that we can develop tools to process them without the need for customization. More complex annotations, however, will require a custom representation. For the example shown in Figure 2, we use the Ontology of units of Measure (OM) [RVAT13] and associate a numerical value and a unit with the body. The Ontology of units of Measure contains further information about om:gramPerCubicmetre like its dimension (a density) or how it can be converted to other units. In general, loading ontologies like the Ontology of units of Measure into the triple store along with our annotations allows us to perform more complex queries.

Sometimes, the body of an annotation should reference another annotation. As an example, we will annotate the occurrences of k in the following theorem:

"Let $k \leq n$ be positive integers. Then $\binom{n}{k} = \binom{n}{n-k}$". Figure 3 visualizes the resulting annotations. The first annotation, :anno0, targets the k in the first sentence. It records that k is universally quantified (i.e. the theorem holds for *all* values of k) and links it to a node :identifier0 that represents the newly introduced identifier. That allows us to link the occurrences of k in the second sentence to :identifier0 (annotations :anno1 and :anno2), which indirectly links them to the declaration of k. Similarly, we can create further annotations to attach additional information about k. For example, :anno3 indicates that *"positive integer"* is something like a type constraint on k.

So far, we have linked annotations via their bodies. We can also link annotations by referencing the same target. For example, we can additionally annotate *"positive integer"* with the semantic concept it refers to (here the entry wd:Q28920044 of the WikiData ontology).

5.3 Discontinuous targets

In Section 5.1 we described selectors for continuous document fragments, but sometimes it is desirable to annotate a discontinuous fragment. For this we have created the **sb:ListSelector**, which combines (lists) selectors for the continuous fragments that make up the discontinuous fragment. For example, in *"every submonoid H of $(\mathbb{Z}, +)$ is ..."*, we might want to annotate the discontinuous fragment *"submonoid of $(\mathbb{Z}, +)$"*, which means that the sb:ListSelector combines selectors for *"submonoid"* and for *"of $(\mathbb{Z}, +)$"*. We do not expect every tool to support discontinuous ranges. Therefore, we attach this list of selectors as a refinement to a selector for the surrounding continuous fragment (*"submonoid H of $(\mathbb{Z}, +)$"*) as shown in Figure 4. A consumer of the annotation can then choose to ignore the refinement and treat the selection as a continuous fragment.

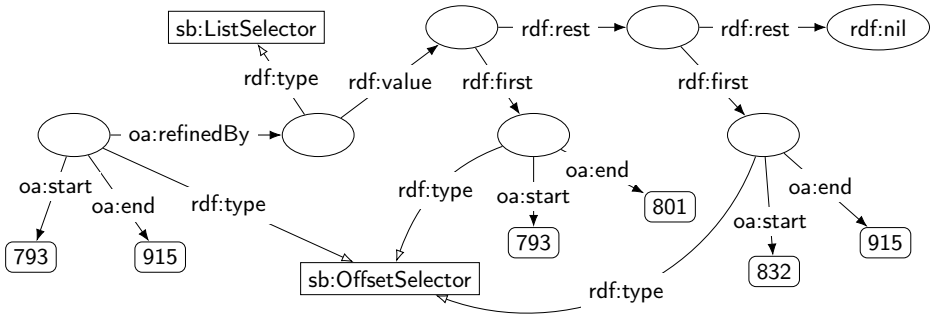

Fig. 4. Example of an sb:OffsetSelector being refined with an sb:ListSelector to select a discontinuous fragment. Using an RDF collection (rdf:rest, rdf:nil, ...) allows us to represent a closed collection despite the open world assumption of RDF.

5.4 Querying annotations with SPARQL

If annotations are stored in a triple store, we can query them with SPARQL queries [HS13]. Usually, we use SPARQL queries to retrieve annotations in a fairly straight-forward way. For example, a hybrid spotter (Section 3) could retrieve all annotations for a particular document.

However, we can also use more interesting and complex queries to look for particular phenomena in our corpus. An example of this would be a query to find papers on group theory that have a theorem which mentions rational numbers:

```
# prefix declarations ommitted for conciseness
SELECT DISTINCT ?paper WHERE {
  # make sure that ?paper is about group theory
  ?paper sb:isBasedOn/^oa:hasTarget/oa:hasBody/rdf:value arxivcat:math\.GR .
  # find theorems in ?paper and look up their offsets
  ?theorem_anno oa:hasBody/rdf:value sbp:Theorem .
  ?theorem_anno oa:hasTarget [
    oa:hasSource ?paper ;
    oa:hasSelector [ a sb:OffsetSelector ; oa:start ?t_start ; oa:end ?t_end ; ]
  ] .
  # Same with mentions of rational numbers (offsets ?q_start, ?q_end)
  ?q_anno oa:hasBody/rdf:value <http://www.wikidata.org/entity/Q1244890> .
  ?q_anno oa:hasTarget [
    oa:hasSource ?paper ;
    oa:hasSelector [ a sb:OffsetSelector ; oa:start ?q_start ; oa:end ?q_end ; ]
  ] .
  # make sure that mention is inside theorem
  FILTER (?t_start < ?q_start && ?t_end > ?q_end)
}
```

Meta spotters (Section 3) could be realized via complex SPARQL queries similar to the one above. For very complex queries, performance can be a concern, given the large amounts of data involved.

6 SpotterBase

The success of our efforts hinges on how easily people of different backgrounds can work with our annotations and create new ones. To support that, we have created a collection of tools and libraries that we call **SpotterBase**. This section discusses some of its features.

6.1 JSON Serialization of Annotations

Sometimes, working with RDF triples is rather inconvenient: a typical annotation consists of 10–20 triples and RDF parsing or a SPARQL endpoint might not be readily available for every task. Furthermore, not everyone is familiar with RDF, which poses a substantial barrier of entry to potential users. To improve

accessibility, we provide a simple JSON format that can be used to import and export annotations. It is a subset of JSON-LD (JSON Linked Data) [Jso], which is a format for representing RDF triples and is recommended by the Web Annotation standard. Many triple stores support JSON-LD and require no additional processing for importing data.

JSON-LD contexts make the JSON content more compact; in addition to the context suggested by the Web Annotation recommendation, we created a SpotterBase JSON-LD context. This context states e.g. that "val" is an abbreviation for `http://www.w3.org/1999/02/22-rdf-syntax-ns#value` and that the value should be interpreted as a URI (and not e.g. a string literal). For example, the annotation of a word as a noun would be exported as:

```
{
  "type": "Annotation",
  "id": ".../arxmliv/2020/1910.06709#spostag.anno.1089",
  "target": ".../arxmliv/2020/1910.06709#spostag.target.1089",
  "body": {
    "type": "SimpleTagBody",
    "val": "http://sigmathling.kwarc.info/spotterbase/universal-pos-tags#NOUN"
  }
},
```

SpotterBase can create additional JSON objects to provide more information about the target and the NOUN part-of-speech tag.

So far, we have mostly used the JSON serialization to export all annotations for a single document, so that they can be visualized and edited in a separate tool. Despite its convenience, the JSON format does not replace the need for a triple store because we need the triple store's querying capabilities.

6.2 Document Narrative Model

Non-STEM NLP tools typically act on plain text and not on HTML documents with large amounts of markup. While converting HTML documents to plain text is fairly straight forward, linking the discovered annotations back to the original document tends to be somewhat tricky. To help with this, SpotterBase provides the **Document Narrative Model (DNM)**: a plain text representation that is linked to the DOM (Document Object Model) of the original HTML document. DNM generation is customizable: depending on the use case, we may ignore certain nodes or process them in a different way. For example, if we want to find identifier declarations with a simple, rule-based approach and have a sentence *"let* $F : \mathcal{C} \to \mathcal{D}$ *be an exact functor"*, then we might want to get the plain text representation *"let MATH be an exact functor"* to make the processing easier. Now we can use simple regular expressions to find potential declarations. Since the DNM links *"MATH"* back to the DOM, we can still retrieve the MathML node for further processing such as to extract the identifier F.

If we develop a hybrid spotter, we could even mark existing annotations in the DNM. For example, if we have already annotated *"exact functor"* as referring

to a mathematical object, we can replace it with a token in the example above, so that we get *"let MATH be an MATH_ OBJ"*.

6.3 Document pre-processing

The document narrative model described in the previous section requires spotter authors to directly use SpotterBase as a library. Alternatively, SpotterBase can also pre-process documents into different formats that are easier to work with. Some of the previous annotation efforts (e.g. [Rab17; Asa+21]) pre-processed the HTML documents by wrapping all words in a node with an identifier to make referencing easier. With SpotterBase, we can do the same thing, except that we attach to every node offset information that allows the spotter to easily create annotations for the original document (using the sb:OffsetSelector described in Section 5.1, from which SpotterBase can then create the corresponding sb:PathSelector).

SpotterBase also allows spotter authors to avoid the trouble of HTML processing altogether with a converter to a JSON format. Essentially, each word in the original document is represented as JSON object of the form

```
{"word": [THE WORD], "from": [OFFSET], "to": [OFFSET]}
```

As with the HTML pre-processing, the offsets allow to annotate the original document. Formulae are represented the same way with a replacement token for [THE WORD], but they have an additional field for the MathML representation. In the future, it might be interesting to explore alternative ways to represent formulae as token sequences.

7 Datasets, spotters and experiences

We have tested the annotation format and SpotterBase by creating several datasets. Concretely, we have imported the following datasets:

1. A *quantity expressions dataset* [Rab17], which was created with a rule-based spotter for finding physical quantities, i.e. pairs of a scalar and a unit.
2. A *formula grounding dataset* [AMA22], which annotates identifier occurrences with a description of what the identifier stands for and, optionally, a source of grounding in the document.
3. A *paragraph classification dataset* [GM20], which contains paragraph classifications (theorem, definition, proof, ...) inferred from markup based on the amsthm LaTeX package. Technically, we re-generated the annotations because the original dataset does not link back into the corpus.

Furthermore, we have implemented a number of prototype spotters. Concretely, we have the following spotters:

1. A spotter for *part-of-speech tags*. As it annotates every word in a document, it produces many annotations (and thus RDF triples), which may stress-test some triple stores.
2. A spotter for *math concepts* that references concepts in the WikiData ontology.

3. A spotter for *variable declarations* that delivers results similar to the one shown in Figure 3. It is a hybrid spotter as it uses the results of the spotter for math concepts to find type constraints.

We have run the last two spotters over 100 000 documents, resulting in roughly 50 million annotations and 800 million triples. After loading them into a triple store, which took a few hours, we can now run SPARQL queries like the one described in Section 5.4 (looking for theorems in papers about group theory that mention rational numbers). The example query takes roughly 350 ms. By changing the query to cover more papers/more common concepts, we determined that the engine can produce roughly 200 results per second for such queries. It should be noted that simpler queries without range comparisons run much more efficiently (e.g. finding all papers that mention rational numbers at all).

As all the datasets are ultimately derived from arxiv.org, they inherit licensing issues. The SIGMathLing project [SML] tries to work around these issues with a data sharing cooperative based on mutual non-disclosure agreements. As some of the created datasets are affected by the same licensing issues, they are only accessible for SIGMathLing members. However, brief excerpts of the annotations are available in a public repository[2]. As the spotters are merely prototypes, the data sets are of limited practical use anyways.

Nevertheless, the prototype spotters allowed us to evaluate and adjust our design decisions. For example, we abandoned an original plan to specify annotation targets using the RangeSelector and FragmentSelector from the Web Annotation specification as it resulted in substantially more triples. The development of the example spotters has also informed and validated the design of SpotterBase features like the document narrative model (Section 6.2), which significantly simplified the spotter implementation.

The ongoing development of a tool for manual annotation by a master's student allowed us to test how well the annotation format works in the context of a web browser. The annotation tool uses the JSON serialization (Section 6.1) to import existing annotations and export the results. While there were some challenges (such as the browser inserting additional nodes into the DOM while loading a document), it was fairly easy to work around them and the sb:PathSelector seems to work quite well. The annotation tool does not generate sb:OffsetSelectors – instead, SpotterBase generates them afterwards from the sb:PathSelectors. In the future, this could be taken further with SpotterBase supporting a variety of selectors, optimized for different applications, and converting between them.

8 Conclusion, ongoing and future work

We have presented a spotter-based approach for accumulating a large collection of diverse semantic annotations to support or enable various semantic services for STEM publications. We use various semantic web technologies to represent, store, and query the annotations. Furthermore, we presented SpotterBase, a set

[2] `https://gl.kwarc.info/SIGMathLing/cicm23-spotterbase`

of tools to create and process annotations. We have tested the setup by importing several existing datasets and creating a few new ones with simple spotters.

The most obvious next step is implementing many different spotters and, afterwards, semantic services like the ones mentioned in the introduction. As spotters are intended to be relatively simple, they can also be an attractive topic for a bachelor's or master's thesis. We tried this in the past with mixed results. A key problem was that a substantial effort of every thesis project was to develop infrastructure for pre-processing, testing, etc. SpotterBase alleviates all of that.

We also plan to grow an ecosystem of tools for working with annotations. Currently, a student is developing a tool for manually creating and editing annotations. While annotations have a standardized target representation, some tasks may require a custom body representation (see also Section 5.2). While a manual annotation tool may therefore support a set of standard representations, it also has to be easily extensible for future annotation tasks. A related tool would be an annotation visualizer that allows us to query the database for interesting annotations and visualize them. For example, we might want to visualize annotations where a spotter disagrees with a test dataset. We are also planning to explore in-document annotations (e.g. via RDFa) as an alternative representation that may be more suitable for certain applications.

In order to have more hybrid spotters and meta spotters, annotations must be available to the community. The easiest way is to share RDF files with the annotations, e.g. as part of the SIGMathLing [SML] effort. We also recently created a SPARQL endpoint which can be used to query some of the annotations. While this is still at a prototype stage, it could become a valuable resource in the future that makes the annotations more accessible. Unfortunately, the prototype SPARQL endpoint is only available to SIGMathLing members for the licensing reasons discussed above.

References

[AK20] Akiko Aizawa and Michael Kohlhase. "Mathematical Information Retrieval". In: *Evaluating Information Retrieval and Access Tasks – NTCIR's Legacy of Research Impact*. Ed. by Tetsuya Sakai, Douglas W. Oard, and Noriko Kando. Springer, 2020, pp. 169–185. URL: https://www.springer.com/gp/book/9789811555534.

[AMA22] Takuto Asakura, Yusuke Miyao, and Akiko Aizawa. "Building Dataset for Grounding of Formulae — Annotating Coreference Relations Among Math Identifiers". In: *Proceedings of the Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, 2022, pp. 4851–4858. URL: https://aclanthology.org/2022.lrec-1.519.

[Asa+21] Takuto Asakura et al. "Miogatto: A math identifier-oriented grounding annotation tool". In: *13th MathUI Workshop at 14th Conference on Intelligent Computer Mathematics (MathUI 2021)*. 2021.

[BR] *brat rapid annotation tool*. URL: http://brat.nlplab.org (visited on 04/06/2023).

[Cas+16] Richard Eckart de Castilho et al. "A Web-based Tool for the Integrated Annotation of Semantic and Syntactic Structures". In: *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 76–84. URL: https://www.aclweb.org/anthology/W16-4011.

[CICM22] Michael Kohlhase and Dennis Müller. "System Description: sTeX3 – A LaTeX-based Ecosystem for Semantic/Active Mathematical Documents". In: *Intelligent Computer Mathematics (CICM) 2022*. Ed. by Kevin Buzzard and Temur Kutsia. Vol. 13467. LNAI. Springer, 2022, pp. 184–188. URL: https://kwarc.info/people/dmueller/pubs/cicm22stexsd.pdf.

[Con] *CoNLL-U Format*. URL: https://universaldependencies.org/format.html.

[FA] *Formal Abstracts*. URL: https://formalabstracts.github.io/ (visited on 02/15/2020).

[Gin+15] Deyan Ginev et al. "KAT: an Annotation Tool for STEM Documents". In: *Mathematical User Interfaces Workshop*. Ed. by Andrea Kohlhase and Paul Libbrecht. July 2015. URL: http://www.cermat.org/events/MathUI/15/proceedings/Lal-Kohlhase-Ginev_KAT_annotations_MathUI_15.pdf.

[Gin20] Deyan Ginev. *arXMLiv:2020 dataset, an HTML5 conversion of arXiv.org*. SIGMathLing – Special Interest Group on Math Linguistics. 2020. URL: https://sigmathling.kwarc.info/resources/arxmliv-dataset-2020/.

[GM20] Deyan Ginev and Bruce R Miller. "Scientific Statement Classification over arXiv.org". English. In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 1219–1226. URL: https://aclanthology.org/2020.lrec-1.153.

[Hal+17] Thomas Hales et al. "A formal proof of the Kepler conjecture". In: *Forum of Mathematics, Pi* 5 (2017). DOI: 10.1017/fmp.2017.1.

[Her+13] Ivan Herman et al. *RDF 1.1 Primer (Second Edition). Rich Structured Data Markup for Web Documents*. W3C Working Group Note. World Wide Web Consortium (W3C), 2013. URL: http://www.w3.org/TR/rdfa-primer.

[HS13] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. W3C Recommendation. World Wide Web Consortium (W3C), Mar. 21, 2013. URL: https://www.w3.org/TR/sparql11-query/.

[HYP] *Hypothes.is*. URL: http://hypothes.is (visited on 04/06/2023).

[Jso] *JSON for Linking Data*. URL: https://json-ld.org/.

[Man+22] Behrooz Mansouri et al. "Overview of ARQMath-3 (2022): Third CLEF Lab on Answer Retrieval for Questions on Math". In: *Experimental IR Meets Multilinguality, Multimodality, and Interaction*. Cham: Springer International Publishing, 2022, pp. 286–310.

[Mil] Bruce Miller. *LaTeXML: A LaTeX to XML Converter*. URL: http://dlmf.nist.gov/LaTeXML/ (visited on 03/22/2023).

[Rab17] Ullrich Rabenstein. "Meaning Extraction and Semantic Services in STEM-Documents – A case study on Quantity Expressions and Units". Master's Thesis. Informatik, FAU Erlangen-Nürnberg, 2017. URL: https://gl.kwarc.info/supervision/MSc-archive/blob/master/2017/urabenstein/Rabenstein.pdf.

[RDF] World Wide Web Consortium (W3C), ed. *Resource Description Framework (RDF)*. URL: http://www.w3.org/RDF/ (visited on 04/05/2023).

[RVAT13] Hajo Rijgersberg, Mark Van Assem, and Jan Top. "Ontology of units of measure and related concepts". In: *Semantic Web* 4.1 (2013), pp. 3–13.

[SML] *SIGMathLing – Special Interest Group on Maths Linguistics*. URL: http://sigmathling.kwarc.info (visited on 12/07/2018).

[Wan+20] Qingxiang Wang et al. "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*. 2020, pp. 85–98.

[Weba] *Web Annotation Ontology*. URL: https://www.w3.org/ns/oa.

[Webb] *Web Annotation Working Group*. URL: https://www.w3.org/annotation/.

[XPa10] *XPath Reference*. 2010. URL: http://www.w3.org/TR/xpath/ (visited on 04/05/2023).