AnnoTize: A Flexible Annotation Tool for Documents with Mathematical Formulae

Lukas Panzer Jan Frederik Schaefer

Department of Computer Science
Friedrich-Alexander-Universität Erlangen-Nürnberg

Abstract

Mathematical formulae are a significant challenge for various services that help us access scientific publications. For example, finding relevant formulae with a search engine is difficult and screen readers struggle to verbalize them in an understandable way. We believe that such services could be improved if more semantic information is available. To automatically extract this information, we typically need manually annotated data sets for evaluation and, potentially, for the training of machine learning models. However, mathematical formulae also present a major challenge for annotation tools, as they cannot be presented well as plain text which is required by most existing tools. In this context, we present AnnoTize, a new annotation tool. It can create, display and update annotations for HTML5 documents and was specifically designed to support the fine-grained annotation of formulae encoded in MathML. As manual annotation can be tedious and time-consuming, AnnoTize offers several features to speed up the annotation process for large annotation tasks.

1 Introduction

The amount of STEM (Science, Technology, Engineering and Mathematics) knowledge is growing rapidly and accessing it efficiently becomes ever more relevant. There are already many helpful services, such as search engines for finding relevant documents or screen readers that help vision-impaired researchers. However, mathematical formulae are particular challenging for these tools. Traditional search engines perform poorly when searching for a formula, which has led to the development of specialized formula search engines (e.g. [PK11]). Similarly, screen readers struggle with reading out formulae in an understandable way. These services could be improved if more semantic information is available. For example, a formula search engine could produce better results if it knows, e.g., whether the e in a formula refers to the Euler number or the electric charge of a proton. Correspondingly, a screen reader could read out the expression $(0, 1)$ much better if it knows whether it refers to an open interval or a point in 2-dimensional space (there is an on-going effort to extend MathML with intent attributes to provide such information [w3cc]). Having such semantic information would also enable the development of other services, e.g. for converting units in a document or allowing a user to evaluate a formula for specific values.

As most documents lack the required semantic information, we can create tools for extracting it, e.g. using natural language processing techniques. The development of such tools requires a set of manually annotated

In: Proceedings of the 14th MathUI Workshop, Cambridge, UK, 06-09-2023, published at http://ceur-ws.org

documents to assess the accuracy and, possibly, to train machine learning models. There are a number of tools for manually annotating documents, but mathematical formulae pose a significant challenge here too: annotation tools typically work on a plaintext representation, but complex formulae cannot be presented well as plaintext.

In this paper, we present AnnoTize[1], a new, flexible tool for manually annotating HTML5 documents with a focus on supporting formulae represented in MathML. AnnoTize is based on a recently proposed RDF-based format for stand-off annotations for STEM documents (see Section 3) in the hope that it becomes compatible with a wider range of tools. AnnoTize aims to overcome some of the limitations of the previously developed (and no longer maintained) annotation tool KAT [DSMT15]. In particular, AnnoTize allows for more fine-grained annotations and does not require a pre-processed document. Furthermore, AnnoTize offers a rapid mode for a more efficient annotation process with fewer user interactions.

Running Example

As a running example, we will annotate identifier declarations in a document. For example, in the sentence *"Let F be a graph"*, we will annotate that the identifier F is declared and that it is universally quantified, i.e. the following statements hold for all graphs F. We will also link later occurrences of F to the declaration. We could also annotate *"graph"* as a type restriction on F, but that would make the example too long. Such information could, for example, be relevant to improve the results of a formula search engine.

All documents used in the examples are from the arXMLiv corpus [Gin20]. ArXMLiv contains HTML5 documents (with MathML for the formulae) that were created from the LaTeX sources of `arxiv.org`.

Overview

First we discuss related work in Section 2. Section 3 provides an explanation of the underlying annotation format. In Section 4, AnnoTize is presented in greater detail, covering the underlying architecture, GUI, and additional features. Lastly, Section 5 concludes the paper and offers an outlook on future work.

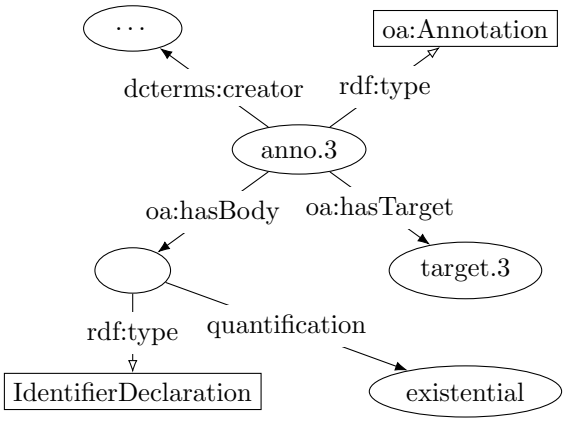
2 Related Work

There are already a variety of different tools for creating annotations. Some of them are general-purpose tools with a focus on annotations for *human consumption*. For example, PDF documents can be annotated with Adobe Acrobat Reader [acr] or Microsoft OneNote [one]. For websites, there are browser plugins like Yawas [Den] and Hypothes.is [hyp]. However, the functionality of these tools is often limited to basic tasks such as highlighting text sections and adding comments. Collaboration among multiple users is often emphasized, but they all lack the ability to handle more complex annotation tasks, such as predefined schemas or relationships between annotations. Moreover, these tools mainly focus on pure text or images, with little or no support for formulas.

We are interested in annotations for *machine consumption*, e.g. to evaluate or train machine learning models. The natural language processing community has created a number of annotation tools for this purpose, such as WebAnno [YGEdCB13], BRAT [SPT+12] and Anofora [CS13]. All of these tools are web-based annotation tools with a focus on user collaboration and flexibility. In BRAT the user can annotate text spans and relations between them. The annotations are visualized through colored boxes while the relationships are visualized with arrows above the text. However, BRAT lacks the ability to annotate more complex schemas, and the annotation process can be time-consuming due to multiple required interactions. WebAnno builds upon BRAT, offering better collaboration support through inter-annotator agreement calculation and support for arbitrary semantic role labeling schemes. It also provides features for more time efficient annotations and a guide for rich semantic tag sets. Anafora has a lightweight, web-based user interface and supports complex schemas.

Annotation tools for natural language processing typically work on plain text. This is a problem for STEM documents as complex formulae cannot be presented well as plain text. Therefore, a few annotation tools were designed to specifically be able to work with formulae. KAT [DSMT15] allows the annotation of HTML documents with formulae represented in MathML. It can be customized for different annotation tasks with a specification file and supports relational annotations, i.e. annotations that reference a different part of the document. KAT offers separate modes for annotating, reading and reviewing the document. Annotations can be exported to a custom format based on the Resource Description Framework (RDF). While KAT has some similarities with our work, in contrast to AnnoTize, KAT can only annotate text fragments on a DOM level and

[1] `https://github.com/rezakul/AnnoTize`




```
{
  "type": "Annotation",
  "id": ".../0909.4913#anno.3",
  "target": ".../0909.4913#target.3",
  "body": {
    "type": "IdentifierDeclaration",
    "quantification": ".../existential
       "
  },
  "creator": "urn:..."
}
```

Figure 1: Sketch of an example annotation as an RDF graph (left) along with its JSON representation (right).

therefore restricts the flexibility of annotation borders. In addition, the annotation process in KAT is slowed down through a lot of user interactions. KAT is no longer maintained. MioGatto [TYAK21] was specifically designed for the task of grounding identifiers. While MioGatto is therefore tailored exactly to the requirements of this task, it prevents reuse for other annotation tasks.

3 Preliminaries: The Annotation Format

A central challenge around the annotation of STEM documents is the lack of a standardized format for storing and sharing annotations. [SK23] attempts to change that by proposing a new, flexible annotation format, which is the foundation of AnnoTize. Our hope is that AnnoTize becomes a central part of an ecosystem of tools supporting this standard.

The annotation format proposed in [SK23] is based on the Resource Description Framework (RDF). RDF triples express statements about resources in terms of subject-predicate-object relationships, which form a directed graph. Figure 1 sketches the RDF graph of an example annotation.

The annotation format follows the recommendations of the W3C Web Annotation Working Group [w3cd]. At its core, an annotation consists of a **target**, describing what should be annotated, and a **body** that carries the information that should be attached to the target. Additionally, an annotation may be linked to some meta information, such as its creator or its creation date.

The format described in [SK23] does not impose restrictions on the structure of annotation bodies to keep the format as flexible as possible. In the example in Figure 1, the annotation body contains a tag indicating that the annotated variable declaration is existentially quantified. For other declarations, more complex bodies can be used. AnnoTize deals with this flexibility through specification files for different body types, which are described in Section 4.4.

The target specification is omitted in Figure 1 for brevity. It would link to the source document and a number of selectors that specify the annotated range of the document. HTML nodes can be specified with XPath expressions and characters in text nodes can be specified with offsets. This allows for very fine-grained annotations without the need to modify the source documents. It is also possible to select discontinuous ranges (which is also supported by AnnoTize), though we have rarely needed this in practice.

The conventional way to interact with RDF triples is to store them in a graph database and query it with SPARQL queries. However, that raises the barrier to adoption of the annotation format in two ways: i) users have to know/learn about RDF and SPARQL, and ii) using a graph database and SPARQL queries is not convenient in every setting. As an alternative, [SK23] suggests a JSON representation of annotations and provides a conversion tool. This JSON representation is based on JSON-LD (JSON for Linked Data) [w3ca], which allows the encoding of RDF triples in JSON. With the use of special contexts, the JSON format should be fairly intuitive for users not familiar with RDF. Figure 1 sketches the JSON representation of an example annotation. AnnoTize works with this JSON representation internally and supports it for the import and export of annotations.

[SK23] describes the annotation format and the motivation behind it in more detail.

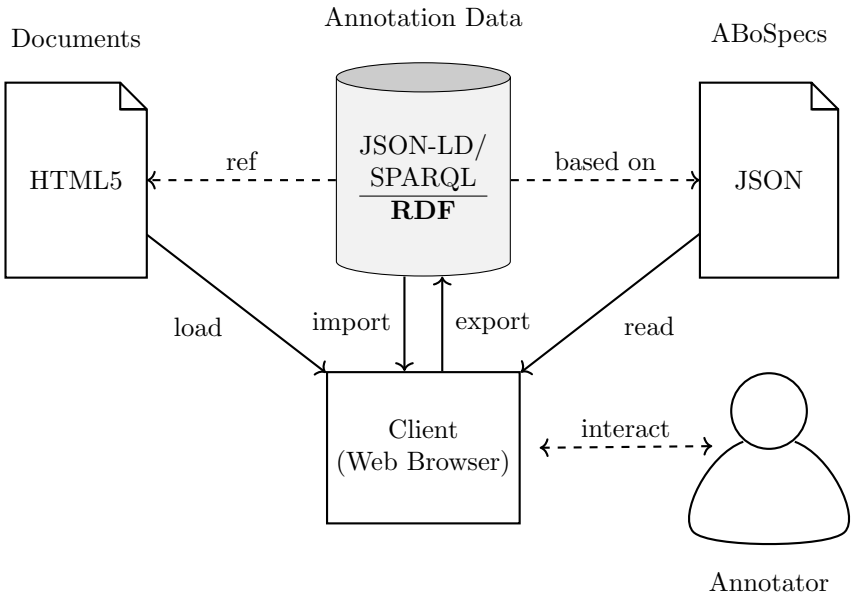


Figure 2: The AnnoTize architecture.

4 AnnoTize

AnnoTize is a JavaScript-based web annotation tool and can be operated either as a stand-alone server or as a browser plugin. It allows annotation tasks on HTML5 documents without requiring preprocessing or compliance with certain rules, apart from being valid HTML documents to ensure that references can be resolved correctly. Mathematical formulae can be represented with MathML, which is supported by all major browsers at the time of writing. It was a priority during the development of AnnoTize that the annotation of MathML fragments is fully supported.

When working with AnnoTize, a user has to first load one or more HTML documents. It can be configured that only certain document sections are displayed for annotation. This can be useful in practice as annotating only a single section per document allows the annotator to cover more documents and therefore potentially annotate a more representative subset of the corpus. Additionally, the user can upload existing annotations for the documents as well as configuration files, such as the annotation body specification, short ABoSpec, (see Section 4.4) or templates (see Section 4.5). The annotations are represented in the JSON-LD based format discussed in Section 3 and can be imported and exported from the user interface. For some annotation tasks, AnnoTize needs further data, like tag sets and tag descriptions, which can also be loaded via JSON-LD files. The JSON-LD files can be created from a SPARQL endpoint with the `SpotterBase` library [SK23]. The advantage of using JSON-LD files is that AnnoTize users do not require access to a SPARQL endpoint. We are considering to additionally support direct interaction from AnnoTize with SPARQL endpoints in the future.

4.1 Creating Annotations with AnnoTize

Creating annotations in AnnoTize is a straightforward process facilitated by a small fixed sidebar on the right-hand side of the document (see e.g. Figure 3). This sidebar contains all the necessary information about the existing annotations.

This section presents the "standard mode" for creating annotations, which is designed to be intuitive and is aimed at smaller annotation tasks and casual users. One drawback, however, is that it requires many user interactions and is therefore relatively time-consuming. In Section 4.5 we present a rapid mode that provides mechanisms to efficiently handle large and complex annotation tasks.

Before the actual annotation work, the desired annotation body specifications and tag sets must be defined and loaded. We will discuss the annotation body specifications in Section 4.4 in more detail and jump right into the actual annotation process.

When a user selects a text fragment, a pop-up window appears, allowing the user to create a new annotation by clicking on it (Figure 3). The selected text fragment serves as the target for the annotation. Figure 3

shows a snapshot of the annotation process for the annotation of identifier declarations. In Theorem 1, the first occurrence F has been annotated as a universally quantified identifier. The variable m is currently selected and can be annotated by clicking on `Annotate`.

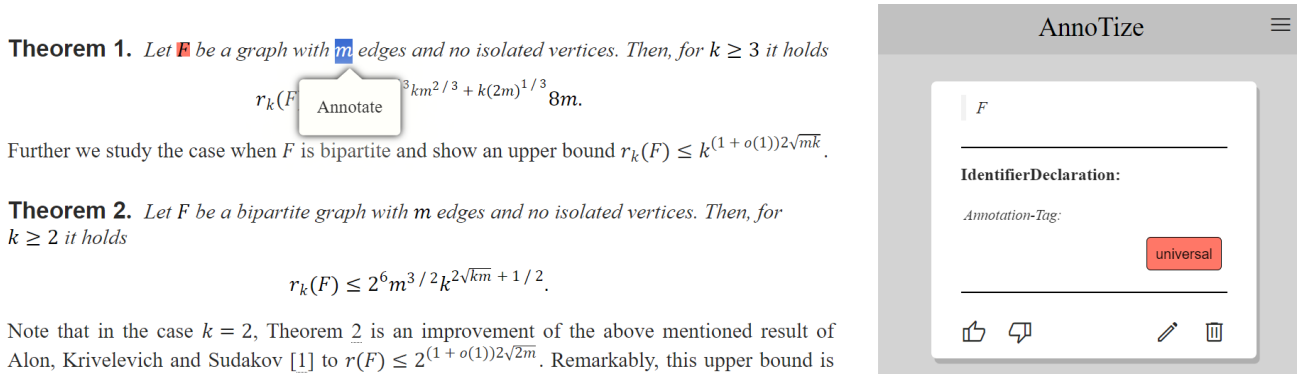


Figure 3: The tool screen for annotating declarations with a pop-up after the user selected some text. The example is taken from the `ar5iv` paper [JP13].

Once an annotation is created, a new entry appears in the sidebar that allows the user to define the body of the annotation. Figure 4 illustrates this by extending the previous example with annotations for identifier occurrences. Concretely, we want to annotate that the k in "$\ldots \leq k^{3\cdot\ldots}$" occurrence of the previously declared identifier k ("*for* $k \geq 3$ *it holds*..."). First, the user selects a body type from a drop-down list. In our example, we select the `IdentifierOccurrence` body. Afterwards, the relevant input fields are displayed, allowing the user to enter the required information. In our case, we have a field to link the occurrence to a previously annotated declaration. The declaration can be selected from a dropdown menu or by clicking on the declaration annotation within the text. The `Save` button becomes active when valid values have been entered for all inputs.

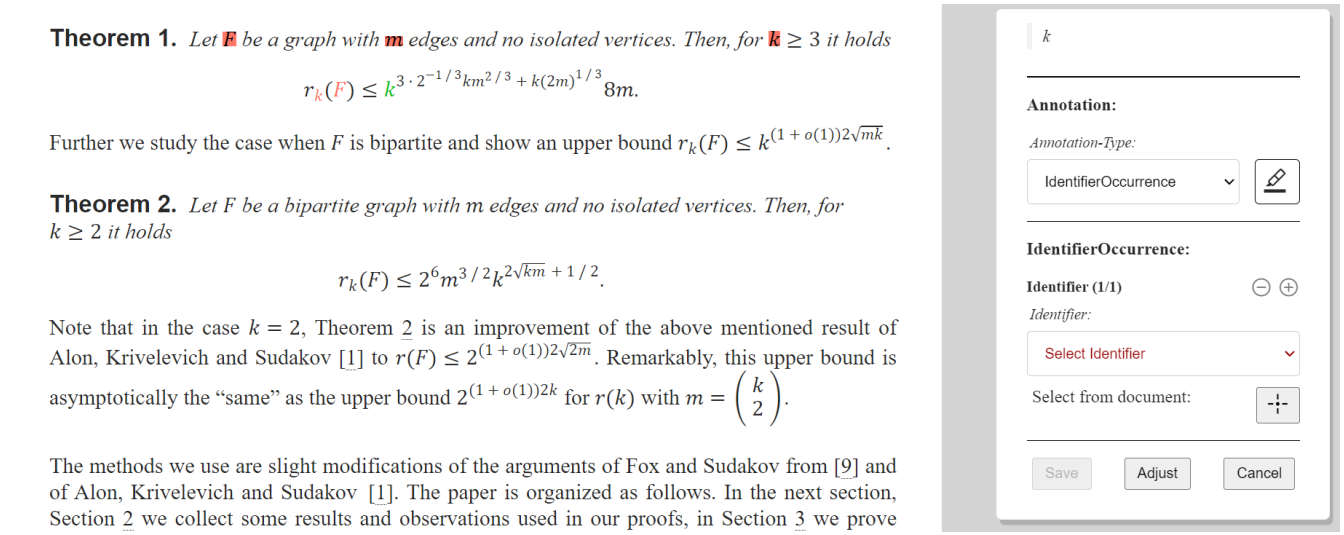


Figure 4: Annotation of an identifier occurrence in the sidebar element.

Some fields in the body can have multiple values. In this case, values can be be added or removed using the $+$ and $-$ buttons associated with the corresponding input field.

When the user hovers over an annotated text fragment, the corresponding entry in the sidebar is highlighted, and, conversely, hovering over the sidebar entry highlights the associated text fragment. Clicking on an annotation triggers a scroll to the corresponding annotation/sidebar element, ensuring visibility. Additionally, if there are references to or from other annotations, hovering reveals an arrow indicating the direction of the reference, along with the label if applicable. Figure 5 illustrates this feature: by hovering over the occurrence of k in Theorem 1, an arrow to the linked identifier declaration appears and the sidebar element is highlighted. Figure 5 also

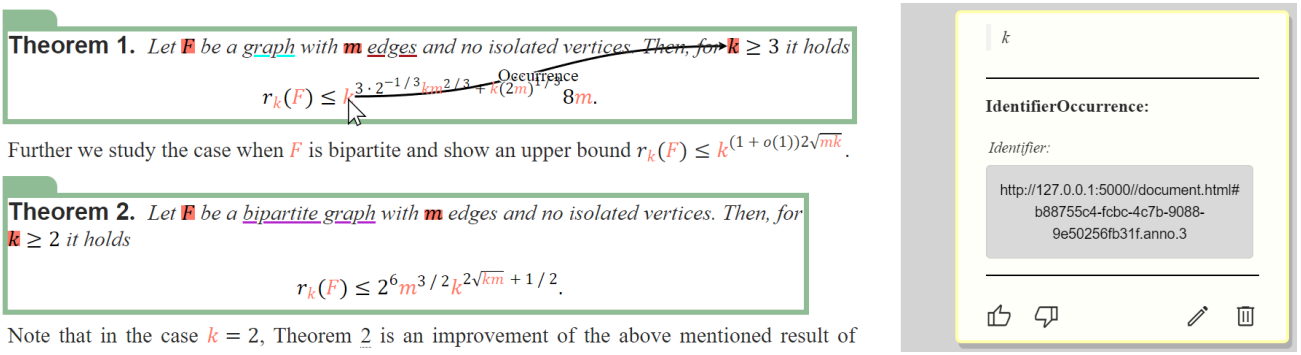


Figure 5: AnnoTize with multiple highlighting styles and the cursor hovering over an annotation to highlight relevant information.

includes other annotations to illustrate the use of different highlighting styles, which we will discuss in the next section.

Saved annotations (including automatically generated ones) can be reviewed with thumbs-up or thumbs-down ratings. Furthermore, they can be flagged as unclear with comments for discussion. This feature is also useful during the annotation process if the annotator is not sure how to handle a particular phenomenon.

4.2 Highlighting Annotations

AnnoTize, like other annotation tools, highlights annotated document fragments. At the moment, the user can choose between 4 highlighting styles: `Marker`, `Colored`, `Underline` and `Rectangle`. `Marker` is a colored box behind the annotated text fragment. To improve the readability for darker box colors, the text color is changed to white if the contrast between the text and box color is below a certain threshold. The W3C content accessibility guidelines are used to determine this threshold [w3cb]. `Colored` changes the text color of the annotated text while `Underline` underlines the text in given color. `Rectangle` creates a rectangle around the text fragment and is intended for bigger fragments like whole paragraphs, where the aforementioned styles would introduce a certain illegibility, especially in combination with other annotations.
The highlighting style can be separately defined for each annotation type. In addition, a specific style can be set for the creator of an annotation. Figure 5 illustrates how different highlighting styles can be combined in a document. The `Rectangle` style is used for the paragraph classifications, while `Marker` is utilized for declarations and `Colored` for identifier occurrences. The `Underline` style is used for type restrictions of declarations. The combination of different highlighting styles provides a clear visual differentiation between different annotation types.

Aside from the highlighting style, colors can convey additional information about each annotation. Each annotation type has a default color, which is initially picked randomly but can be changed by the user. Additionally, tags are also associated with their distinctive color. When an annotation type incorporates a tag within its body, the color attributed of the tag can take precedence over the annotation type's default color. Similarly, in cases where concepts reference other annotations, the color of the referenced annotation can supersede the annotation type's default color. While this policy works reasonably well, we plan to refine it in the future for more complex use cases.

4.3 Adjustment of the Annotation Target

AnnoTize allows the user to adjust the annotation target during the editing process. Users have the option to reselect the text fragment, add additional document ranges to create a discontinuous target, or fine-tune the left and right borders of the annotation target. This feature proves useful in various scenarios. When correcting automatically generated annotations, the ability to reselect the text fragment can be utilized to ensure accurate targeting. Fine-tuning the left and right borders is particularly beneficial when annotating mathematical formulae.

Consider the example of annotating the expression $\sqrt{mk}$ from Figure 4, which is represented in MathML as

```
<msqrt...><mrow...><mi...>m</mi><mo...></mo><mi...>k</mi></mrow></msqrt>
```

The annotation format allows for precise targeting using nodes or character offsets (see Section 3), enabling precise selection of any conceivable fragment. However, manual selection in the browser presents challenges, such as recognizing invisible nodes like the invisible multiplication operator between the identifiers. To assist users, the selected boundaries in the XML structure are shown in the adjustment screen (Figure 6).

(a) Selection of identifier (b) Selection of radicand (c) Selection of square root

Figure 6: The adjustment screen for different annotations in the example formula $\sqrt{mk}$.

Another difficulty arises when the user wants to annotate the entire radicand mk (e.g. `<mrow>` as a target) in one annotation. Since the browser does not allow direct selection of the square root symbol, selecting only the radicand instead of the entire square root (e.g. `<msqrt>` as a target), may be difficult. However, with the adjustment functionality, users can easily refine their selection. The adjustment dialog (Figure 6) shows the selected html elements (black) along with the surrounding elements for context (gray).

4.4 Preparing Annotation Tasks: ABoSpecs and more

When researchers create a new annotation task, a key design decision is the format of the annotation body, i.e. determining the precise information that should be contained in the body. While some types of annotation body are commonly used (e.g. a single tag), we often need customized bodies. In Section 4.1 the annotation bodies contained tags and references to other annotations. Other complex annotation bodies can be, for example, quantities (a scalar and a unit) or the grounding of math identifiers as used in [TYAK21] (a description, a math type and an arity).

AnnoTize supports JSON-based specification files for annotation bodies, which we call **ABoSpecs** (**A**nnotation **Bo**dy **Spec**ifications). These files are created once for the annotation task and later imported by the annotators. Concretely, an ABoSpec specifies the fields that annotation bodies should include. Five different field types are currently available:

1. **Tag**: a tag from a tag set
2. **Reference**: a reference to another annotation
3. **Number**: a numerical input value
4. **Text**: a text input for one or a few words
5. **TextBox**: a free text input for longer phrases (typically less relevant as we focus on annotations for machine consumption)

Each field can have additional constraints. The tag field can be linked to a tag set consisting of predefined or dynamically created tags. For the reference field, we can specify the role of the referenced annotation and optionally restrict the referenced annotation types. For the number field, it is possible to specify a range of allowable values with minimum, maximum and step values. The text field allows the specification of a regular expression that the input must match. Finally, the text box can specify a maximum length the input must adhere to.

Additionally, for each input the number of possible values can be specified. For optional fields, the minimum number of values can be set to zero. Similar, we can require at least one, at most one, or exactly one value.

The five field types cover a wide range of possible annotation tasks, but are not exhaustive and could be easily extended. For the future, it would be desirable to introduce a plugin functionality that allows users to implement their own custom fields as needed.

Aside from the fields, every ABoSpec have a name that uniquely identifies it and, optionally, a default highlighting style and color. For example, the ABoSpec for identifier occurrences is:

```
{
    "type": "ABoSpec",
    "name": "IdentifierOccurrence",
    "description": "The occurrence of an identifier that was declared
       elsewhere",
    "style": "colored",
    "fields": [{
        "type": "reference",
        "name": "Identifier",
        "role": "source",
        "label": "occurrence",
        "referencedTypes": ["IdentifierDeclaration"],
        "number": {
            "atleast": 1,
            "atmost": 1
        }
    }]
}
```

Besides the creation of ABoSpecs preparatory tasks include the creation of tag sets and the selection of suitable documents. Both of these tasks fall beyond the tool's current scope.

4.5 Faster Annotations with Rapid Mode

As mentioned earlier, creating annotations in the standard mode involves a substantial number of user interactions. Other authors of annotation tools already noted that the annotation interface and mainly the mouse interactions have a great impact on the annotation speed [YGEdCB13, SPT+12]. Apart from the time and potential monetary loss, a lower annotation speed also results in a less pleasant experience. Tools like WebAnno [YGEdCB13] and BRAT [SPT+12] implement features that limit the number of necessary user interactions while annotating. AnnoTize implements a similar concept with the rapid mode. The example annotation of an identifier occurrence in Section 4.1, for example, required in the normal mode first the selection of a text fragment, followed by five mouse clicks to create the annotation body. Complex bodies would require even more interactions. rapid mode aims to remove unnecessary interactions and provides features to speed up repetitive tasks. In the following paragraphs, we will discuss some of the features offered by the rapid mode.

Annotation templates: Templates help with repetitive annotation tasks. A template fixes the annotation type and can define default values for input fields. During the annotation task, a new annotation is created according to the selected template. The user can create up to five different templates and switch between them either in the sidebar or by pressing the numbers 1 to 5 on the keyboard. If the template sets values for all required fields, the annotation can be automatically saved immediately after creation. For fields that reference other annotations, it is possible to automatically reference the most recently selected or created compatible annotation. This gives the user a fast way of creating annotations with references. Of course, references and other field values can still be modified as needed for each annotation.

Keyboard-based annotation: As Yimam et al. [YGEdCB13] discussed keyboard-based annotation has a huge speed advantage over mouse-based annotation. The navigation in the annotation body can be entirely done by using the keyboard. As mentioned above, templates can be switched with the number keys. As soon as a new annotation is created the first input field is focused, allowing the user to immediately enter the field value. It is possible to switch between the input fields with the tab key. During tag selection, pressing a key selects the first tag that starts with the corresponding character and pressing the key again selects the next one.

Adding annotations without a pop-up: As soon as the user selects a text fragment, a new annotation is created. The keyboard focus will be transferred to the first input field of the annotation body. This saves the user a click on the "Annotate" pop-up.

Target normalization: Currently the only supported normalization feature is the removal of trailing spaces. Most annotations end on word boundaries and do not contain spaces at the end. When double-clicking on a word,

many modern browsers like Firefox, Google Chrome or Edge will automatically include the space after the word. Although it is possible to disable this behavior in the settings of some browsers, AnnoTize provides an option that removes trailing whitespaces without altering the default behavior of the browser. If the annotation task includes the selection of single words, a double-click can noticeably increase the annotation speed. We plan to support more normalization modes in the future, for example only allowing for the selection of whole paragraphs for tasks like paragraph annotation. Similarly, the annotation of complete figures, table cells or formulae may be desirable for some tasks.

Without the pop-up and with the removal of trailing spaces, annotations in the example task can be conveniently created by simply double-clicking on the identifier in the text. The annotation process can be further sped up with templates for declarations (one with the existential quantification tag and one with universal quantification tag), as well as a template for identifier occurrences which automatically refers to the last interacted declaration.

For example, consider the task of annotating all declarations (3) and occurrences (14) of identifiers in Theorem 1 from Figure 3. In the standard mode, a text fragment must be selected for the annotation of each identifier, resulting in a total of 17 selections. In addition, 108 mouse clicks are required for the bodies. With the rapid mode and the three templates described above, only two keystroke are required per identifier, one to select the corresponding template for the declaration and one to select the template for occurrences. Additionally, a double-click is required to select each identifier. This reduces the number of interactions to 17 double-clicks and 6 keystrokes.

5 Conclusion and future work

We have described AnnoTize, a flexible web-based annotation tool for HTML5 documents that supports the annotation of formulae. It is based on the annotation format that was proposed in [SK23] and we hope that it can become a key part of the envisioned ecosystem of tools built on that format. We have tested AnnoTize with different types of annotation bodies. Aside from identifier declarations and occurrences, we have annotated part-of-speech tags, paragraph classes (theorem/definition/...), quantity expressions (like *"5 kg"*) and references to mathematical concepts. In particular, an early version of AnnoTize is used in an on-going master's thesis to evaluate the results of a tool for finding quantity expressions.

Our next goals are to

1. **Improve support for open tag sets:** While small, closed tag sets are well supported, some tasks, such as the annotation of mathematical concepts (*"graph"*, *"complex number"*, ...) requires working with a large set of tags (that might come from an existing ontology). Adding new tags or selecting existing ones from a large set is still relatively inefficient with AnnoTize.
2. **Explore applicability to other XML-based document formats:** While we started out with the goal of supporting math/STEM documents in the HTML5 format, it may be possible to adapt AnnoTize to support document formats used in other fields. In particular, we plan to explore the application to corpora in the TEI (Text Encoding Initiative) format. We are also exploring the possibility to integrate AnnoTize into the WissKI system [wis], which is used to manage knowledge e.g. about historical artifacts using semantic web technologies.
3. **Improve extensibility with plugin infrastructure:** AnnoTize already different field types in ABoSpecsas well as a number of highlighting styles for annotations. While this is sufficient for a wide range of annotation tasks, the options offered may in some instances not fully meet the requirements of a particular task. We plan to make AnnoTize more versatile by providing a plugin infrastructure that makes it easy to extend its capabilities for specific annotation tasks.

References

[acr] Adobe Acrobat Reader DC. `https://acrobat.adobe.com`. Accessed: 2023-07-10.

[CS13] Wei-Te Chen and Will Styler. Anafora: A web-based general purpose annotation tool. In *Proceedings of the 2013 NAACL HLT Demonstration Session*, pages 14–19. Association for Computational Linguistics, 2013.

[Den] Laurent Denoue. Yawas. https://github.com/ldenoue/yawas. Accessed: 2023-07-10.

[DSMT15] Ginev Deyan, Lal Sourabh, Kohlhase Michael, and Wiesing Tom. Kat: an annotation tool for stem documents. In *Mathematical User Interfaces Workshop*, 2015.

[Gin20] Deyan Ginev. arxmliv:2020 dataset, an html5 conversion of arxiv.org. `https://sigmathling.kwarc.info/resources/arxmliv-dataset-2020/`, 2020. SIGMathLing – Special Interest Group on Math Linguistics.

[hyp] Hypothes.is. `http://hypothes.is`. Accessed: 2023-07-07.

[JP13] Kathleen Johst and Yury Person. On the multicolor ramsey number of a graph with m edges, 2013.

[one] Microsoft OneNote. `https://www.microsoft.com/de-de/microsoft-365/onenote/digital-note-taking-app`. Accessed: 2023-07-10.

[PK11] Corneliu C. Prodescu and Michael Kohlhase. Mathwebsearch 0.5 - open formula search engine. In *Wissens- und Erfahrungsmanagement LWA (Lernen, Wissensentdeckung und Adaptivität) Conference Proceedings*, September 2011.

[SK23] Jan Frederik Schaefer and Michael Kohlhase. Towards an Annotation Standard for STEM Documents. In *Intelligent Computer Mathematics (CICM) 2023*, LNAI. Springer, 2023. in press.

[SPT+12] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. brat: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107. Association for Computational Linguistics, 2012.

[TYAK21] Asakura Takuto, Miyao Yusuke, Aizawa Akiko, and Michael Kohlhase. MioGatto: A Math Identifier-oriented Grounding Annotation Tool. In *13th MathUI Workshop at 14th Conference on Intelligent Computer Mathematics (MathUI 2021)*, 2021.

[w3ca] JSON-LD 1.1: A JSON-based Serialization for Linked Data. `https://www.w3.org/TR/json-ld11/`. Accessed: 2023-07-26.

[w3cb] Web Content Acessibility Fuidelines . `https://www.w3.org/TR/WCAG20/`. Accessed: 2023-07-25.

[w3cc] MathML intent. `https://w3c.github.io/mathml-docs/intent-explainer/`. Accessed: 2023-08-10.

[w3cd] Web annotation working group. `https://www.w3.org/annotation/`. Accessed: 2023-07-08.

[wis] Wisski - a scientific communication infrastructure. `https://wiss-ki.eu/`. Accessed: 2023-08-10.

[YGEdCB13] Seid Muhie Yimam, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. WebAnno: A flexible, web-based and visually supported system for distributed annotations. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 1–6. Association for Computational Linguistics, 2013.