ALAP: Accessible LaTeX Based Mathematical Document Authoring and Presentation

Ahtsham Manzoor
Lahore University of Management Sciences
Lahore, Pakistan
15030043@lums.edu.pk

Safa Arooj
Lahore University of Management Sciences
Lahore, Pakistan
safa.arooj@lums.edu.pk

Shaban Zulfiqar
Lahore University of Management Sciences
Lahore, Pakistan
shaban.zulfiqar@lums.edu.pk

Murraiyam Parvez
Lahore University of Management Sciences
Lahore, Pakistan
16100070@lums.edu.pk

Suleman Shahid
Lahore University of Management Sciences
Lahore, Pakistan
suleman.shahid@lums.edu.pk

Asim Karim
Lahore University of Management Sciences
Lahore, Pakistan
akarim@lums.edu.pk

Abstract

Assistive technologies such as screen readers and text editors have been used in past to improve the accessibility and authoring of scientific and mathematical documents. However, most screens readers fail to narrate complex mathematical notations and expressions as they skip symbols and necessary information required for the accurate narration of mathematical content. This study aims at evaluating a new Accessible LaTeX Based Mathematical Document Authoring and Presentation (ALAP) tool, which assist people with visual impairments in reading and writing mathematical documents. ALAP includes features like, assistive debugging, Math Mode for reading and writing mathematical notations, and automatic generation of an accessible PDF document. These features aim to improve the LaTeX debugging experience and make it simple for blind users to author mathematical content by narrating it in natural language through the use of integrated text to speech (TTS) engine. We evaluated ALAP by conducting a study with 18 visually impaired LaTeX users. The results showed that users preferred ALAP over another comparable LaTeX based authoring tool and were relatively more comfortable in completing the tasks while using ALAP.

Keywords

Accessible LaTeX; Accessible Math; Text to Speech; Navigation; Assistive Debugging; Computers; Visually Impaired; PDF Accessibility

ACM Reference Format:
Ahtsham Manzoor, Safa Arooj, Shaban Zulfiqar, Murraiyam Parvez, Suleman Shahid, and Asim Karim. 2019. ALAP: Accessible LaTeX Based Mathematical Document Authoring and Presentation. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland Uk.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3290605.3300734

1 Introduction

In the recent years, a substantial amount of research has been conducted with the aim to improve the accessibility of existing tools and development of new tools for assisting blind users in creating scientific and mathematical documentation. Assistive technologies such as screen readers and text editors perform reasonably in terms of continuous text [23], however, these technologies tend to skip mathematical notation and symbols in their speech output, and more importantly do not support mathematical content creation and editing that is accessible to the blind. Previous implementations in this regard are also largely limited to simple mathematical equations and symbols or reading of mathematical content rather than its creation and editing (e.g., Lambda [2], AudioMath [3]).

Our initial survey to understand the needs of science students with visual impairments in Lahore (Pakistan) revealed that two popular editing tools that are being used by the blind scientific community are MS Word and LaTeX (TeXlipse) paired with renowned screen readers like JAWS (Job Access with Speech) [19] and NVDA (NonVisual Desktop Access) [17]. However, the widely used screen readers like JAWS that perform successfully in terms of basic computer usage and textual data, have failed to fully support accessible mathematical narration in the resulting PDF files. These tools skip mathematical notation and symbols in their speech output. Moreover, working in LaTeX involves coding and debugging which can be daunting for blind individuals. Popular screen readers do not support LaTeX systems fully and are restricted to basic navigation and editing support without understanding LaTeX syntax. ChattyInfty [27], on the other hand, understands LaTeX and makes its editing via speech accessible; however, it does not assist with LaTeX debugging which is unavoidable while working in LaTeX. The key inaccessibility challenges in LaTeX are:

(1) Screen readers are incapable of understanding LaTeX and processing complex mathematical notations and symbols.
(2) LaTeX based errors are not easy to debug and no current system supports assistive debugging in LaTeX [?].

After identifying our key inaccessibility challenges, we held interviews and surveys in order to evaluate the following research questions:

(1) Is the available technology (screen readers, editors) accessible enough to author and publish documents comprising advanced mathematical content independently?
(2) How LaTeX debugging impacts the user's performance in authoring scientific document?
(3) How does the narration of mathematical content in natural language facilitate the visually impaired users?
(4) Does automating the process of accessible PDF generation is a step forward towards more inclusive technology?

In this work, we report on the design, development, and evaluation of an accessible LaTeX system, called ALAP (Accessible LaTeX-based Authoring and Presentation), for accessible mathematics. ALAP includes the following features that contribute towards our attempt to create a unified solution for Scientific and Mathematical documentation by blind users:

(1) A text-to-speech (TTS) engine that understands LaTeX and supports speech based creation and editing of mathematical documents
(2) An accessible debugging platform in LaTex that would navigate through errors and narrate them to the user.
(3) A Portable Document Format (PDF) generator that enables accurate narration of the resulting files containing complex symbols and notations.

2 Literature Review

Based on our aforementioned contributions, we have divided our research into the following three subsections:

Constructing an Accessible Editor for Authoring Scientific and Mathematical Documentation

Mathematics is a two dimensional, non-linear language that requires equally complex representation. Whereas the simpler equations can be written in their linear substitute form, the intricate ones require adequate two dimensional representations [6]. Over the years computer researchers have sought and developed various tools to author and narrate Mathematical symbols and notation on electronic devices in order to aid the blind community. Some of the state-of-the-art solutions include Lambda [2], an educational mathematics tool for visually impaired students that help them manipulate and work with basic school level mathematics. Although, the idea behind the creation of lambda is quite distinctive, this tool has limited support for complex mathematical notations and equations.

AudioMath [3] is another significant contribution towards creating online documents, including mathematical information, accessible to the blind but it fails to provide support for complex notations. Furthermore, the project is limited to mathematical document reading and does not support creation. The MathSpeak [12] project is also an appreciable effort towards making mathematics accessible. However, it is meant for users who want to convert their work, ultimately, to Braille. "TalkMaths" is another practical tool that allows blind users to author mathematical content by giving verbal input [26].

The idea of creating mathematical documents by dictation has not been seen before and this makes "TalkMaths" a novel contribution. Unfortunately, creation of mathematical documents via verbal input has its limitations. For instance, this tool works at a satisfactory level for individuals with English as their native language [26]. Moreover, the user study conducted for this tool indicates that even though users liked the concept of "TalkMath", they were unable to perform as well as expected [26]. The tasks involved in the evaluation also include basic mathematical equations including linear and quadratic forms instead of large fractions and integrals as commonly seen in the scientific context. Thus, it is evident that existing systems providing accessibility support to the blind community either do not allow mathematical document authoring in entirety or are too cumbersome in usage.

Accessibility of Existing LaTeX Editors with Focus on Assistive Debugging

Before extending TeXlipse to assist blind users, we tested the existing LaTeX editors to identify their limitations and inadequacies. All editors were tested with same test cases using both JAWS and NVDA. Shortcut keys of both these screen readers failed to work with "Overleaf" [18]. Additionally, some of the basic LaTeX commands were mispronounced.

For example, "\sin" was read out as "backslash sin" instead of "backslash sine". Screen readers also couldn't differentiate between "\sin" and "\sinh" or "\tan" and "\tanh". These issues often led to ambiguity which diminished the overall experience of editing. "ShareLaTeX" [20] had the exact same problems with an additional flaw of just narrating the characters and not the complete word while typing.

Similarly, "TeXlipse" was tested which revealed that shortcut keys of both screen readers (JAWS and NVDA) are supported by it. The LaTeX document could be compiled easily using "Ctrl + S" but with no feedback. Hence, to ascertain the potential errors or no errors, user must manually switch to the problem window on each build. Finally, to fix an error, user must switch back to the editor and then navigate to the particular line which is evidently a very challenging task [1].

ChattyInfty [27] was also tested to understand and determine its accessibility support. It works extremely well in terms of narrating and editing mathematical content. It pronounces mathematical notations in a human understandable form. It appends phrases like "sub end" and "frac end" to indicate the end of subscript and fractions respectively. This feature helps clarify the structure of the equation for users with visual impairment.

However, we encountered a few primary complications while using ChattyInfty [27], which we plan to resolve in our system to ensure complete accessibility while creating and editing LaTeX based documents. First, previewing documents in PDF using this tool is rather inconvenient, as user must first export the file as LaTeX, which will also compile the document and save the output as .dvi file. Then this .dvi file must be converted to a PDF file. This whole process is extremely tedious and needs to be repeated after every change. Second, ChattyInfty [27] lacks support for debugging the LaTeX code. ChattyInfty provides no auditory feedback upon compilation of a document, hence, leaving the user in a state of confusion concerning the status of the document.

However, it does disable the "Show DVI file" option as a subtle indication of a failed compilation. The only possible way for a user to spot and fix an error is to listen to the entire document line by line. Table 1 shows a brief comparison of existing LaTeX system from the perspective of editing, navigation and debugging when paired with JAWS/NVDA.

In order to mitigate the challenge of debugging in LaTeX, the SZS-Editor [13] has provided a Simplified Log of Compiling which includes two features called A Go-to-line and an Auto-completion LaTeX-code functionality. This feature allows SZS-Editor to summarize the error log in a few lines. The Go-to-line feature received a score of 4 in the standard System Usability Scale [13]. Our attempt at assistive debugging is also inspired by this functionality which will mitigate the challenges found during the process of error debugging.

One effort to assist visually impaired programmers is the implementation of a C sharp tool by Tran et al which also makes the use of Text to Speech (TTS) engine [24]. This TTS based C sharp tool attempts to make debugging accessible. It does so by informing the user upon the occurrence of a compilation error and reads out the error description, along with mentioning the file and the specific line number [24]. Also, predefined shortcuts allow the user to navigate between lines of error conveniently.

Our LaTeX based system also uses these features to make debugging in LaTeX more accessible. However, this tool fails to simplify error reporting, something that is vital to LaTeX based errors. In fact, errors in LaTeX are sometimes long and this can lead to confusion, especially for visually impaired users who struggle to obtain information from their surroundings [8].

The major difference between LateX debugging and code debugging is that, coding compilers offer breakpoint functionality to debug the source code line by line in real time. On the contrary, LaTeX debugging is asynchronous i.e. it will compile the whole document once the build command is executed. Therefore, after execution, the user has to deal with the errors by narrating their description and line number. In our system, however, the errors being narrated are simplified by excluding unnecessary descriptions and warnings. Moreover, redirecting the cursor focus to the start of the erroneous line greatly helped the users in fixing the errors. [10] The results section shows the positive impact of this design decision.

Generating an Accessible PDF File

Like authoring and presenting Mathematics in an editor, narrating the resulting Mathematical file is a problematic task due to the wide spectrum of symbols and expressions [21]. The narration can also be ambiguous due to the confusion of grouping the expressions [16]. Prominent screen readers including JAWS and NVDA provide limited support to mathematical content. While JAWS is able to read out some complex mathematical equations, they have to be formatted

Tool	Editing & Navigation	Debugging	Keys Compatibility (JAWS/NVDA)
Overleaf	✓	X	X
ShareLaTeX	✓	X	X
TeXlipse	✓	✓*	✓
TeXmaker	X	X	✓
TeXworks	✓	X	✓
TeXStudio	✓	X	X
Authorea	✓	X	X*
ChattyInfty	✓	X	✓

Table 1: A table showing support for editing, navigation and debugging of existing LaTeX systems. ✓*: system supports partial debugging accessibility. X*: editing and navigation is partially supported i.e. JAWS commands only work with plain text but not with Latex commands

according to the MathML format and require the MathPlayer plugin. Added to this, JAWS is limited to Windows based platforms.

NVDA also reads out equations formatted in MathML well and so does the Voice Over utility in MacOS based systems. However, reading mathematical content in other forms including that in PDF's is unsatisfactory in nature. In addition to this, when it comes to editing mathematical documents, these readers coupled with text editors lack full support for editing. ChromeVox screen reader [21] attempts to narrate chromebooks to the blind users, its existing solution only works for high school and undergraduate mathematics.

ALAP not only ensures accessible mathematical content creation but also enables narration of the resulting accessible PDF file in its entirety without skipping any symbols and expressions. Although, the idea behind most of the aforementioned technologies are novel, none of them provides a unified solution for authoring, presenting and narrating Scientific and Mathematical documentation.

3 System Design And Implementation

To establish possible requirements for the accessible LaTeX system, we conducted a survey and interviewed our targeted audience (blind users). Participants were asked to specify the problems that they encounter while authorizing mathematical documents, how they fix errors that occur while creating LaTeX-based documents and which other tools they use for creating scientific documents. In addition to other aspects, the interview focused on collecting the design interaction suggestions, ideas and system feedback medium. Users were also asked to state the limitations of existing screen readers while working with and authoring scientific documents.

The overall design of ALAP was based on the results of this initial user research and it is aimed towards corresponding to our goal of creating accessible LaTeX based Mathematical-editing tool. We decided to use the open source Eclipse plugin called TeXlipse [22], which was formerly used for generating LaTeX based documents. By opting for an open source solution, we hope to build a community around our product so that a larger group could extend it. We integrated Microsoft's text to speech (TTS) engine [14] with TeXlipse for communicating information to the user. The TTS API also provides extended customizable functionality including control options like pause, resume and ready. It also allows switching between male & female voice and gives control over play speed.

The detailed implementation of the developed system is as follows:

Assistive Debugging

ALAP uses one lookup table for special characters and one for the LaTeX commands with their human narration. Upon receiving an input, the TTS in our system transforms the input into its corresponding value found by searching in its respective table. For instance, if the TTS is given '}' as an input, it will speak "closing curly bracket". It uses "CTRL+S" to save and build the LaTeX source. If there are errors, two events get invoked respectively: 1) The TTS speaks out the first error message along with the line number. 2) The cursor is moved to the start of the erroneous line. This feature completely removes the problem of navigating within errors and provides coverage for both compile time and runtime errors. On successful build, our system speaks "no error found".

Compile time errors occur prior to building the source when a token e.g. '}' is missing. Runtime errors occur upon building the source when, for instance, a LaTeX command (e.g. `\section{}`) has been entered incorrectly or syntax structure has not been followed properly by a user. Both types of errors are shown in the "Problems" window at the bottom of the editor with line numbers. For runtime errors, if a user has misspelled or used a command not supported by the current imported packages, ALAP embeds the phrase "Either misspelled or missing package" with the actual error message.

ALAP ignores warnings and unnecessary error descriptions and provides real time support for both compile and run time errors. This offers users a chance to fix errors on

Key	Description
CAPS+O	Turns ON TTS(ON by default)
CAPS+Q	Turns OFF TTS
CAPS+S	Narrates the document depending upon the verbosity level
CAPS+P	Pauses the TTS prompts.
CAPS+R	Resumes the paused state of the TTS
CAPS+E	Sets the TTS in ready state even if it is in paused state.
CAPS+W	Sets verbosity level of TTS to word.
CAPS+C	Sets verbosity level of TTS by character.

Table 2: A table of shortcut keys for TTS settings

Key	Description
CAPS+G	Speaks line number of the cursor
CAPS+H	Enables and disables cursor mode
CAPS+I	Speaks from current cursor position to end of line and "No further text" if text ends
CAPS+J	Speaks from immediate word at which the cursor is currently appearing to end of line.
CAPS+K	Speaks from start of the line till cursor position
CAPS+L	Speaks complete line where the cursor is at.
CAPS+M	Speaks from immediate word to end of the document.

Table 3: A table of shortcut keys for Editing & Navigation

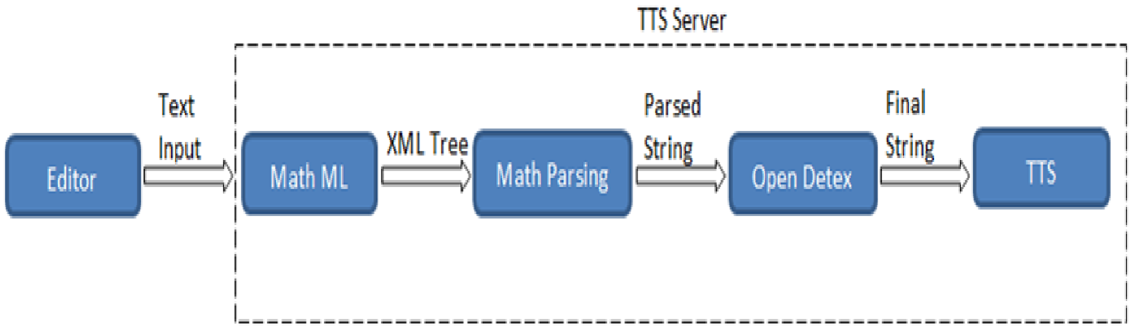

Figure 1: Overall system design flow for Math Mode.

the go instead of waiting till compilation hence, resulting in a convenient way to work with LaTeX.

Warnings don't block build process and ignoring them removes the confusion between actual error and warning. However, concrete investigation of LaTeX warnings can be done appropriately. A user can also switch between multiple errors upon pressing CTRL+> and CTRL+< to navigate to the next and previous error respectively. When the errors are resolved, the user can successfully build the LaTeX source code and subsequently generate a PDF document.

We assist our users in debugging and navigation by providing the following features:

(1) The users can switch between two verbosity levels (character and word) for TTS narration. At character level, the TTS speaks out the input character by character while at the word level, TTS speaks out the input word by word. The character verbosity level is essential to read out punctuation and special characters. We have provided a number of TTS settings along with the two verbosity levels shown in Table 2. These settings are both supported by action icons and shortcut keys in order to ensure an accessible interaction design.

(2) When the cursor mode is enabled with the verbosity level set to word, TTS speaks out each word on pressing space while writing or on directing to any word. Similarly, in the character mode, TTS speaks out each input character (alphanumeric \special). Additionally, our system detects and notifies if a user starts writing in caps or jumps to a new line. However, we have minimized the risk of enabling caps lock ON by disabling caps lock, when it is pressed with our system defined keys. The TTS will start speaking on selection of a particular piece of code depending upon the verbosity mode. We have implemented multiple shortcut keys shown in Table 3 to maximize the accessibility of source navigation for blind individuals. Some of our editing and navigational features are common with typical screen readers like JAWS and NVDA as they are fundamental for navigation purpose.

Yap Viewer [28] is configured in our system that gives flexibility of split window to show the PDF document. It shows the output of the source code on run-time by pressing "CTRL+S" and supports reverse search feature. This feature completely removes the problem of run time error, which occurs when we recompile the same PDF document that is already opened in the external PDF viewer for instance Acrobat, Foxit etc.

Math Mode

Math mode is implemented to make mathematics accessible through LaTeX. We have used MathML [11] utility which takes error free LaTeX syntax of math equations as input and returns an XML tree object as output. The XML object is then used to parse the mathematical equations according to the rules defined for each mathematical operator. Finally, OpenDetex [5] utility is used to remove the TeX constructs, which results in a final output string which is then further passed to the TTS for narration. At this stage, all the content rendered on output PDF along with mathematical equations and expressions will be spoken out in natural language. Thus, Math mode feature will not only eliminate the need of external screen readers, e.g. JAWS, NVDA etc., to read documents but also make advanced and complex mathematical content more accessible by precisely conveying the essential information to the user.

We have implemented rules for each type of mathematical operator. A LaTeX command is identified by the keyword "\". For instance, in case of \frac; the source code of its numerator and denominator will be extracted and according to the predefined rule, firstly the numerator gets parsed, then \frac is looked up in the LaTeX commands table and finally the denominator gets parsed. All the subparts of an expression are concatenated, resulting in a final output string, which is then narrated by the TTS. In this way the equation, for instance, `$\frac{3}{4}$` will be spoken as "fraction numerator 3 over denominator 4 end Frac" by the integrated TTS.

Similarly, if we have multiple mathematical expressions in a single equation, for instance, `$\sum_{n=1}^{\infty} 2^{-n} = \frac{n}{3}$`, the system will first truncate and process the summation rule, then the power rule and finally the fractional part. Each type of sub expression will be parsed in its own block according to the pre-defined rule and the output of each block will then be concatenated in an orderly manner. The expected narration of the above-mentioned expression will be "summation underscore n is equal to 1 end sub raise to power infinity end power 2 raise to power minus n end power is equal to fraction numerator n over denominator 3 end Frac". Hence, ALAP is intelligent enough to process any type of mathematical equation namely, integration, differentiation, logics, trigonometric operators, limits, summations, partial differentiation, matrices, tables, pseudo code algorithms etc.

We have treated the most commonly used environments for writing mathematical expression like `'$'`, `'$$'`, `'\['`, `'\('`, `'\begin{displaymath}'`, `'\begin{eqnarray*}'`, `'\begin{equation}'`, `'\begin{math}'` etc.

The figure 1 describes the overall system design. This design is being followed to convert the LaTeX source into human narration of mathematical content without losing any information unlike other screen readers. CAPS + D command is used to speak the source code in Math mode.

Accessible PDF

The purpose of this feature is to automate the process of accessible PDF generation. Users can turn this feature on by using CAPS + A or by pressing the action icon. If this feature is turned on, upon compilation ALAP will parse the LaTeX code and notify the user in case of any violation of PDF accessibility standards as defined by WCAG 2.0 [25]. ALAP alerts the user of potential violation by narrating the warning along with the line number. For instance, if the user tries to use the watermark in the document, our system will speak out waning message to not to use watermark as defined by the WCAG 2.0 standards. To ensure that users abide by all the recommended guidelines, they are not allowed to build the code in case of violation. The highlight of "Accessible PDF" is the automatic mechanical tagging of math equations with descriptive narration that eliminates the need to manually write the tooltip or alternative text. This will allow most mainstream screen readers e.g. JAWS, NVDA to narrate the mathematical content in correct and precise descriptive form.

4 Experiment Design

Participants

We conducted the study with 18 completely and partially blind computer individuals (Male = 16, Female = 2). The average age of participants was 19.6 (SD = 2.5). Among the participants, 5 were pursuing bachelors and the rest were still in high school. All participants were LaTeX users with varying expertise level.

A pre-experiment questionnaire was conducted to determine the participant's education, background (sciences or others) and previous exposure to screen readers (JAWS, NVDA) and LaTeX. The average experience of screen readers among participants was 5.8 years (SD = 2.8).

All participants had at least six months experience of LaTeX for writing technical documents. They were all able to create a basic LaTeX document comprising of moderately advanced mathematical content, independently.

Set-Up

The study was conducted in a computer lab and the computers used during the experiment were similar with respect to operating system, hardware and environment. The essential tools for the study like JAWS, NVDA, TeXlipse, video recorder and ALAP were pre-installed on each computer.

As all participants were previous users of TeXlipse, they were well acquainted with its interface and shortcut keys. But since the participants were unfamiliar with ALAP, a full day (4 hours) tutorial was held to familiarize the participants with ALAP and acquaint them with shortcut keys required for navigation, editing and other TTS settings. They were also briefed about the ALAP functionalities and its usage. After the briefing and a 20-minute QA session, they were asked to perform small tasks, to help them get familiar with the tool and eradicate the novelty factor. Before beginning this exercise, users were communicated to acquire help from the research team or each other, if they get stuck somewhere. By the end of the tutorial, all participants were well accustomed with ALAP and were able to use it independently. This familiarization with the lab and testing environment established the tone for the experiment and reduced any ineptness they might have faced while using ALAP for the first time.

Tasks:

The testing session kick started on the next day and lasted for three days. The participants were randomly divided into three groups and each group was invited on a different day for the test. Before starting the experiment, the participants were given a briefing of tasks they needed to perform. Participants were also given time to adjust their screen readers (volume, speed, voice) and a choice between NVDA & JAWS. Surprisingly, all participants preferred JAWS over NVDA. The participants were required to perform a total of 8 tasks with both TeXlipse and ALAP. The first task was to simply write a continuous text, the second one was related to debugging, the next 5 tasks were mathematical expressions and the final one was related to creating a accessible PDF with mathematical content.

The details of these tasks are following:

(1) Write the following continuous text: "Computer Science is the study of the theory, experimentation and engineering that form the basis for the design and use of computers".

(2) Debug and fix the existing compile/run time error in the source code. The following two equations were used to generate compile and run time error respectively.
   (a) `$\x^{2$`
   (b) `$\sumation_{n=0}^{5}f(x)$`
(3) Write LaTeX code for the equation: $\sqrt{x^2+1}$
(4) Write LaTeX code for the equation: $\frac{x}{y} \geq \frac{x^2}{y}$
(5) Write LaTeX code for the equation: $\sum_{n=1}^{\infty} n^2$
(6) Write LaTeX code for the equation: $\frac{\partial Q}{\partial t} = \frac{\partial s}{\partial t}$
(7) Write LaTeX code for the equation: $\sin^2(2\theta)+\cos^2(3\theta) = 1$
(8) And finally generate PDF and identify the equation $\sqrt[3]{x^3+1}$ using JAWS or NVDA

The purpose of task 1 was to evaluate the ease of typing continuous text and ability to fix spelling mistakes using ALAP, task 2 was designed to evaluate assistive debugging feature for both compile and run-time errors. Task 3-7 was used to assess the effectiveness and accuracy of ALAP's feature called Math Mode. These tasks cover a range of topics. Moreover, these tasks determine the extent to which ALAP assist blind individuals to work independently with mathematical content in order to publish or understand scientific documents using platform. Task 8 was particularly designed to determine and compare the quality of JAWS narration of a PDF document, having mathematical expressions, generated using TeXlipse and ALAP.

Procedure

The pre-experiment questionnaire took about 10 minutes per participant. For the second part of study, all the users were given 5 min per task to complete it. However, the time for task 4 was 7 min due to its complexity. The participants performed all the tasks with both ALAP and TeXlipse, hence to avoid partialities, order of the tasks and whether they used ALAP first or second was balanced through random shuffling. The video recorder was turned on right before the test started. Video recorder was used to extract the statistical values of the evaluation metrics like, task time, error rate, and error recovery rate for later analysis.

Since all participants were LaTeX users; the tasks were narrated in mathematical form i.e. in English and they received no assistance for LaTeX source code. Participants were not provided any human assistance during the test to avoid any biased effect, except narrating the task again if requested and helping with ALAP/TeXlipse shortcut keys required to complete a certain task.

Users were not interrupted while taking test and the time consumed by each user was also recorded for later analysis. Participant's performance was measured on four metrics, (a) completion time per task, (b) error rate and (c) error recovery rate (d) Number of uncompleted tasks.

The time per task started when the invigilator had finished narrating the question and ended when user had successfully finished the task without any syntactic/semantic error. Later, participants verified each task by listening to it in math mode using CAPS+D which was not possible in TeXlipse combined with JAWS. Error rate was the number of errors, whether syntactic or semantic, left in an equation once the user felt he/she has finished it. Error recovery rate was the ability of the user to fix those errors in the constrained time. In case of absolute inability of a user to finish a task in the given time, the task was marked incomplete and the user was instructed to move to the next one.

The final step of the study was the post experiment questionnaire and a short interview to take feedback from the users. The questionnaire was used to get quantitative feedback on the usability, ease to learn, debugging support and other various aspects of ALAP. Whereas, the interview comprised of open-ended questions which encouraged participants to talk freely about the shortcomings, suggestions and features of ALAP they liked. The post experiment interview session facilitated us in collecting a decent and formal qualitative feedback.

Results

To analyze the results, we used task completion time, error rate, recovery rate and reported the data insights semantically. We found that, participants were faster and more accurate in completing the tasks using ALAP. The participants had a more enjoyable experience with ALAP as they were less frustrated and were more confident in working with mathematical content in a natural language.

Task Completion Time

For TeXlipse the average task completion time was 21m 2s, (SD = 5m 55s) and for ALAP the average time was 21m 13s, (SD = 5m 7s). The difference between these averages is minor, but as obvious from Figure 2, the number of participants whose performance enhanced while using ALAP, clearly, surpassed TeXlipse. P-10 was still in high school and had 3 years of experience with JAWS. Hence, P-10 struggled while working with TeXlipse using JAWS. P-10 performed better with ALAP, but still he consumed the most time in both scenarios.

66.66% of participants performed better while using ALAP and the performance of the remaining 33.33% was better with TeXlipse. The reason why those participants performed well with TeXlipse was mainly due to the concurrent behavior of JAWS. TTS is designed linearly i.e. it narrates every single character that user types as oppose to JAWS. Another major reason for the less time consumed per task while using TeXlipse was the inability to fix semantic mistakes. Once the code had successfully compiled and there was no error in the "Problem Window", participants had no way of listening to the PDF, identifying their semantic mistake and fixing it. Since while using TTS, participant could listen to the PDF in descriptive form and then fix semantic errors, this time was also included in task completion time.

ALAP also succeeded in achieving a lower frequency rate of incomplete tasks. Participants were unable to complete a total of 13 tasks while using TeXlipse. In ALAP, this count dropped to 7 and the reason was the assistive debugging feature and the Math Mode. Due to the automatic redirection of the cursor at erroneous line, participants had to spend less time locating the error. In case of TeXlipse, the tedious process of switching between the editor and problem window caused a lot of participants to run out of time.

Error Rate

Error rate was the count of errors in the Problem Window, once the user was confident that he/she has successfully completed the task and compiled the document. The number of semantic mistakes was also added to the count, once the document compilation was successful but output was incorrect. The average error rate (ER) for participants was 5.33 (SD = 3.4) in case of TeXlipse and using ALAP, the average ER was 4.94 (SD = 2.66).

There was no significant difference between ALAP and other tool in terms of committing mistakes.

However, the total ER in TeXlipse is higher than ALAP. The largest number of errors were committed during task 4 due to its complexity level i.e. 21 in TeXlipse and 23 in ALAP. Similarly, in task 6, the ER was minimum i.e. 10 in TeXlipse and 7 in ALAP.When participants did the task 4, nine participants missed one of the braces, 3 entered extra braces, and 3 of them entered the fraction command wrongly.

Three of the participants did not make any mistake in this task for both ALAP and TeXlipse. From the figure 4, we can see that, the ER of 13 participants using TeXlipse is higher than or equal to ALAP. The reason why some participants performed better while using ALAP might be due to the linear eloquence of TTS.

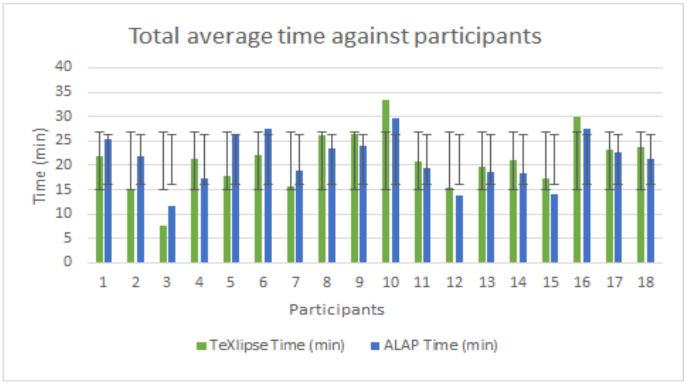

Figure 2: Average time against participants with ALAP and TeXlipse.

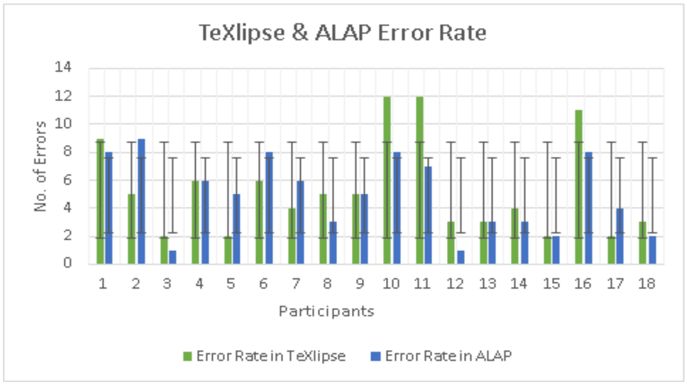

Figure 3: Error rate of ALAP and TeXlipse.

We also observed this situation many times and users realized their typos only after they had compiled the code, which often led to errors. Participants might feel confident while writing swiftly based on their individual skills.The only case where users were unable to recognize their errors was the semantic mistakes while working with TeXlipse. However, it was never the case with ALAP since the descriptive narration was precise enough for users to identify their semantic mistakes.

Error Recovery Rate

The error recovery rate (ERR) is the total number of errors that user fixed under stipulated time limit for each task. The average ERR for participants was 3.83 (SD = 1.82) in case of TeXlipse and 4.44 (SD = 2.12) using ALAP. Participants were more efficient in resolving errors using ALAP than TeXlipse. The participants were successful in resolving 69 errors out of 96 (71%) without ALAP and 80 errors out of 89 (89.88%) with ALAP. The ERR of 3 participants were higher in TeXlipse than ALAP. 5 of the participants were successful in resolving all the errors occurred while completing the tasks both with ALAP and TeXlipse (Figure 3). This is due to the reason that these participants committed very few errors (figure 3, 4) and were successful in resolving them within the stipulated time.

However, the remaining 7 of the participants were successful in achieving higher ERR while using ALAP. The maximum difference between RR and ERR against a participant was 7 while using TeXlipse. Similarly, the maximum difference was 2 in the case of ALAP.

All the measures show that most of the participants were efficient and successful in resolving and completing their tasks due to assistive debugging and Math Mode features. Participants also used Math Mode for correcting semantic mistakes. For instance, 4 people wrote $\sqrt{x}^2 + 1$ instead of $\sqrt{x^2+1}$. On building the source code compiler didn't throw any error since both the expressions were correct and produced the results accordingly. While using ALAP, participants were able to identify the mistake by using Math Mode

(CAPS + D). Since JAWS cannot read out the mathematical expressions accurately, hence participants were unsuccessful in identifying such semantic errors while using TeXlipse. The ERR cross validated the assistive debugging feature and Math Mode feature that helped users to resolve errors efficiently and assisted in the overall completion of the tasks.

Accessible PDF

Task 8 was particularly designed to evaluate the accessible PDF feature of ALAP. Participants were asked to generate two separate PDF files by using both TeXlipse and ALAP, but with same mathematical equation. ALAP automatically tags math equations with descriptive narration which enables other screen readers like JAWS, NVDA to correctly read these equations. JAWS feedback for the ALAP generated accessible PDF was, "cube root inside radical x raise to power 3 end power plus 1 end radical". Because of this extensive and precise feedback, 100% of participants were able to identify this equation accurately. Whereas, for the PDF file generated using TeXlipse, JAWS narration was "3x + 1". This description was not even remotely close to the original equation since JAWS skipped out most of the necessary information while reading the PDF document. Hence, not even a single participant was able to identify it.

Debugging

Task2 was particularly designed to evaluate the debugging experience for both TeXlipse and ALAP. The average completion time for participants was 234 seconds (SD = 60.72) in TeXlipse and 192 seconds (SD = 67.04) for ALAP.

The trend lines of both TeXlipse and ALAP show that debugging experience with ALAP was much better than TeXlipse (Figure 5). P16 was unable to complete the task in stipulated time using TeXlipse and his time was the highest (295 s) among all the participants while using ALAP. The error recovery rate in this particular task (Task 2) was also 100% while using ALAP and 89% with TeXlipse.Overall, participants took longer to fix errors in TeXlipse than ALAP; the reason being the required manual navigation to, and inside the "Problem Window", to listen to the error description.

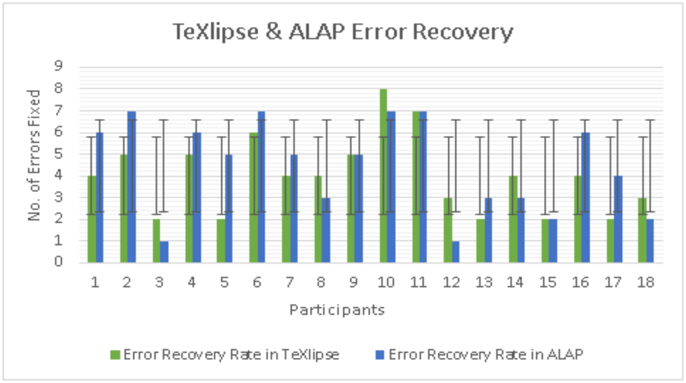

Figure 4: Error recovery rate of ALAP and TeXlipse.

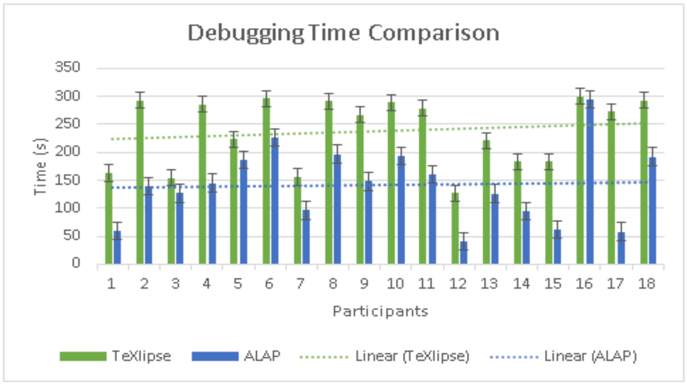

Figure 5: Analysis of debugging time with ALAP and TeXlipse.

To fix the error, participants had to switch back to the editor and re-navigate to the erroneous line. This long hectic process resulted as higher task completion time. It is not surprising that participants performed better with ALAP in contrast to TeXlipse as providing the assistive debugging feature was one of the main goals of ALAP.

Technology Acceptance Model Test

We used the Technology Acceptance Model (TAM) [7] using a 7-point Likert scale ranging from "strongly disagree" to "strongly agree" in order to measure the usability and acceptance of our system (Table 4). All 18 of our participants were required to fill the questionnaire at the end of their survey. The results show that the users gave high scores in all the questions in the questionnaire, contributing to adequately high mean scores in all the six usability matrices. The highest scores were received in the System Accessibility (SA) instrument with the least standard deviation because all the participants acknowledged the benefits of the Math Mode and Mathematical narration in the resulting PDF file.

Evidently, the less experienced LaTeX users faced some challenges while using ALAP however all of them appreciated our features like assistive debugging, Math Mode, and automatic generation of accessible PDF.Mainly, they felt daunted by errors and debugging and were not able to immediately get acquainted with the assistive debugging feature of ALAP, thus contributing to a lower mean score for Perceived Ease of Use (PEU).

5 Qualitative Results

Overall, the majority of participants were motivated to use ALAP. They appreciated the editing environment and were willing to use it on a regular basis. The entire session, including the interviews was recorded, the interaction details noted and then coded by two researchers for further analysis. This section summarizes the qualitative results of the post task interviews.

Almost all participants claimed that the TTS was fluent, easy to understand and proficient in narrating all forms of LaTeX input. P7 stated, *"Other LaTeX based systems have no* *speech support, deeming them inaccessible for blind individuals. Your system provides speech support which is a great thing. The TTS used in your system reads out errors clearly which helps in debugging."* Most of the participants were former JAWS and NVDA users. Therefore, inadvertently, they were comparing our text-to-speech engine with JAWS or NVDA. They claimed that JAWS narration is concurrent with the cursor position while occasionally, our TTS lags. Some participants (P2 and P17) also suggest that *"another TTS voice, namely "Eloquence" would also have been a good choice for your system"*.

Most of the participants thoroughly appreciated the accessibility of debugging provided by the system. For instance, P12 said, *"Error handling was good and easy because of the redirection feature. The best thing is that I did not have to go through each and every line manually to find the source of error"*. P12 qualitatively compared our system with existing LaTeX editors by saying, *"Previously, I had to navigate through lines to find the error manually and error descriptions contained unnecessary details. Your system allows automatic detection of errors and makes debugging a lot simpler"*. While 14 out of 18 participants found ALAP easy to use during the session, all 18 of them agreed that they would be able to easily learn its usage in a short time. 10 of the participants were convinced that the assistive debugging feature would help them with independent coding in LaTeX. P2 stated that *"Math Mode is really a powerful tool to work with mathematics as it precisely conveys the essential information and it will help us in creating mathematical documents in future"*. 7 of the participants gave us the suggestion to speak complete line when we move the cursor up and below in the editor. They might want to map the behavior of our TTS with JAWS or NVDA.

Despite the few issues faced by some participants due to novelty factor, all participants were able to acknowledge that the features provided in ALAP had the capability to benefit the scientific community. P15 commented, *"That's very exciting work. There is a clear need for such a system especially in scientific fields."*

Item	Mean score ± (SD)
Perceived Ease of Use (PEU)	5.09 ± 0.59
Perceived Usefulness (PU)	5.73 ± 0.35
Attitude (A)	5.89 ± 0.99
Behavioral Intention (BI)	5.61 ± 0.27
System Accessibility (SA)	6.54 ± 0.26
Self-Efficacy (SE)	5.31 ± 0.31

Table 4: TAM Model Measures

6 Discussion & Limitations

The lack of support for accessible mathematical based document authoring is indeed an important issue that needs to be targeted. Our initial user research and literature review, validate this gap and stress on the importance of the kind of an accessible LaTeX based system that we developed.

Around the world, a number of blind users have been trying to find an optimal editor that would support scientific documentation. Some pair Microsoft Word [15] and other basic editors with screen readers while others rely on Braille [4]. Only a very few among them are able to use LaTeX because the existing LaTeX editors are devoid of built-in support for blind users.

We have tried to address such fundamental challenges by making LaTeX based document authoring more accessible, especially when it comes to debugging. Based on the results, our proposed system has been successful in facilitating the blind professional and students in scientific documentation. It has also been able to promote LaTeX and motivate blind individuals to pursue STEM [9] fields. The participants seemed determined to continue and increase the use of LaTeX, as it is now relatively more accessible to them due to the features provided by ALAP.

We would also like to highlight some design decision which we took while implementing our system and results showed that they work well.

(1) Speech based output can immensely help blind users in editing. However, this speech should be customizable (tone, speed, etc.).
(2) Blind users do not like ambiguity and therefore should be informed about every micro-level action on the editor as we used TTS for each operation either on the editor or in the whole process
(3) Blind users require keyboard shortcuts for ease of control and want to be notified through speech when the state is changed using shortcut commands.
(4) In terms of debugging, automatic error narration and redirection to the exact location of the error assists blind users and this implementation can make any editor more usable.
(5) Narrating the mathematical content precisely in natural language impacts a lot in making mathematics accessible.
(6) Automatic generation of accessible PDF document is very helpful for both students and researchers with and without any visual impairment as it eliminates user overhead and ensures correctness.
(7) Sighted users can disable the TTS engine with a simple shortcut key while still enjoying other features of ALAP (e.g. debugging support).

When evaluated, our system was accepted by most participants. The results show that users, who have to write scientific documents on daily basis, found our system extremely useful, this also highlights one of the limitations of our study. In Pakistan, finding blind computer users highly proficient with LaTeX was not an easy task; we had to rely on the available LaTeX users. Based on the results, we are confident that our proposed system is effective in serving visually impaired individuals to write a wide variety of math documents, which none of the existing tools has made possible. Our integrated TTS is able to read out all the symbols and notations unlike the existing readers, which skip or incorrectly pronounce a lot of mathematical and scientific content. This would mean that the entire process of working in LaTeX, starting with the code and ending with a PDF file is accessible to the blind users.

The study was conducted with students having different levels of exposure to LaTeX i.e. non-proficient to proficient. ALAP's experience with professional researchers or experts could have been different. In addition to this, the user's proficiency with JAWS and previous experience with TeXlipse could be limited and that must have impacted the overall results and is again a threat to the validity of our results.

7 Future Work

We hope to extend this system in the future. The current version of our system is only supported as a windows desktop-based application. We have already started to convert it into an online tool such that the user does not have to install any additional packages and can use the accessible LaTeX system on any available browser without caring about any operating system. Moreover, we aim to deeply investigate and build a strategy on how to handle LaTeX warnings more appropriately and to determine their impact on the authorization of scientific documents.

8 Conclusion

The existing LaTeX based systems and screen readers lack support for accessible mathematical based document authoring and presentation for visually impaired people. ALAP makes LaTeX based mathematical documents accessible to those with blindness. ALAP includes features like assistive debugging, Math Mode, and automatic accessible PDF generation to assist blind users in the process of authoring and presentation of scientific documents. Our system was tested by visually impaired people. All the participants acknowledged the fact that the system made mathematical documents authoring more accessible to them. The statistical results show that participants were positive about ALAP and they would be interested in continuing to use the tool. The participants found it easy to work with accessible mathematical and scientific documentation through the use of ALAP.

9 Acknowledgment

This work was supported by a grant from Ignite, a national research and development fund. A number of other individuals also worked on this project at various stages. In particular, we acknowledge the contributions of Muhammad Umar Hayat, Muhammad Qasim Hunain, and Aiza Anjum Chaudhry.

References

[1] Catherine M Baker, Lauren R Milne, and Richard E Ladner. 2015. Structjumper: A Tool to Help Blind Programmers Navigate and Understand the Structure of Code. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. ACM, 3043–3052.

[2] Alistair DN Edwards, Heather McCartney, and Flavio Fogarolo. 2006. Lambda:: A Multimodal Approach to Making Mathematics Accessible to Blind Students. In *Proceedings of the International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 48–54.

[3] Helder Ferreira and Diamantino Freitas. 2005. Audiomath: Towards Automatic Readings of Mathematical Expressions. In *Proceedings of In Human Computer Interaction International*. HCI International.

[4] American Foundation for the Blind. 2016. What Is Braille? Retrieved Jan 02, 2019 from http://www.afb.org/info/living-with-vision-loss/braille/what-is-braille/123

[5] Brew Formulas. 2015. Open Detex. Retrieved Jan 02, 2019 from https://github.com/pkubowicz/opendetex

[6] Chandrika Jayant. 2006. A Survey of Math Accessibility for Blind Persons and An Investigation on Text/Math Separation. *Seattle: University of Washington* (2006).

[7] William R King and Jun He. 2006. A Meta-Analysis of the Technology Acceptance Model. *Information & Management* 43, 6 (2006), 740–755.

[8] Ivan Kopecek and A Jergová. 1997. Programming and Visually Impaired People. In *Proceedings of the World Computer Congress*, Vol. 98. ICCHP, 365–372.

[9] Melanie LaForce, Elizabeth Noble, Heather King, Jeanne Century, Courtney Blackwell, Sandra Holt, Ahmed Ibrahim, and Stephanie Loo. 2016. The Eight Essential Elements of Inclusive STEM High Schools. *International Journal of STEM Education* 3, 1 (21 Nov 2016), 21. https://doi.org/10.1186/s40594-016-0054-z

[10] Ahtsham Manzoor, Murayyiam Parvez, Suleman Shahid, and Asim Karim. 2018. Assistive Debugging to Support Accessible Latex Based Document Authoring. In *Proceedings of the International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 432–434.

[11] MathML. 1998. Mathematical Markup Language. Retrieved Jan 02, 2019 from https://www.w3.org/Math/whatIsMathML.html

[12] MathSpeak. 2017. What is MathSpeak? Retrieved Jan 02, 2019 from http://www.gh-mathspeak.com/examples/grammar-rules/

[13] Giuseppe Melfi, Thorsten Schwarz, and Rainer Stiefelhagen. 2018. An Inclusive and Accessible LaTeX Editor. In *Proceedings of International Conference on Computers Helping People with Special Needs*. Springer, 579–582.

[14] Microsoft. 2012. Microsoft Speech Platform. Retrieved Jan 02, 2019 from http://msdn.microsoft.com/en-us/library/jj127449.aspx

[15] Microsoft. 2018. Microsoft Word. Retrieved Jan 02, 2019 from https://www.w3.org/TR/WCAG-TECHS/pdf.html

[16] Emma Murphy, Enda Bates, and Dónal Fitzpatrick. 2010. Designing Auditory Cues to Enhance Spoken Mathematics for Visually Impaired Users. In *Proceedings of the International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 75–82.

[17] NVDA. 2006. NV Access. Retrieved Jan 02, 2019 from https://www.nvaccess.org/

[18] Overleaf. 2012. Real-Time Collaborative Writing and Publishing Tools With Integrated PDF Preview. Retrieved Jan 02, 2019 from http://www.overleaf.com/
[19] Freedom Scientific. 1995. Freedom Scientific"BLINDNESS SOLUTIONS: JAWS. Retrieved Jan 02, 2019 from http://www.freedomscientific.com/Products/Blindness/JAWS
[20] ShareLaTeX. 2014. ShareLaTeX, The Online LaTeX Editor. Retrieved Jan 02, 2019 from http://www.sharelatex.com/
[21] Volker Sorge, Charles Chen, TV Raman, and David Tseng. 2014. Towards Making Mathematics a First Class Citizen in General Screen Readers. In *Proceedings of the Web for All Conference*. ACM, 40.
[22] Texlipse Sourceforge. 2017. "TeXlipse" Homepage - LaTeX for Eclipse. Retrieved Jan 02, 2019 from http://texlipse.sourceforge.net/
[23] study.com. 2003. Continuous and Non-Continuous Texts. Retrieved Jan 02, 2019 from https://study.com/academy/lesson/continuous-non-continuous-texts-definitions-comparison-examples.html
[24] Dat Tran, Philip Haines, Wanli Ma, and Dharmendra Sharma. 2007. Text-To-Speech Technology-Based Programming Tool. In *Proceedings of the WSEAS International Conference on Signal, Speech and Image Processing*. World Scientific and Engineering Academy and Society (WSEAS), 173–176.
[25] W3C. 2018. Make PDF Files Compliant With Web Content Accessibility Guidelines 2.0. Retrieved Jan 02, 2019 from https://www.w3.org/TR/WCAG-TECHS/pdf.html
[26] Angela M Wigmore, Eckhard Pflugel, Gordon JA Hunter, James Denholm-Price, and Martin Colbert. 2010. TalkMaths Better! Evaluating and Improving an Intelligent Interface for Creating and Editing Mathematical Text. In *Proceedings of International Conference on Intelligent Environments*. IEEE, 307–310.
[27] Katsuhito Yamaguchi, Toshihiko Komada, Fukashi Kawane, and Masakazu Suzuki. 2008. New features in Math Accessibility With Infty Software. In *Proceedings of International Conference on Computers for Handicapped Persons*. Springer, 892–899.
[28] YAP. 1991. Yet Another Previewer. Retrieved Jan 02, 2019 from http://tex.imm.uran.ru/tex/yap.html