Math Information Retrieval with Wildcards in the Context of Pen-based Interfaces

CAMILO MUNOZ, University of Waterloo, Canada, 2019

ABSTRACT

Thanks to the advance in mobile and touch screen devices, handwritten input has gained more popularity among users. When considering mathematical input, however, handwritten math interfaces have to deal with new problems and issues not found in natural language. A popular area of interest that deals with math formulae recognition is math information retrieval (MIR). MIR starts with a user query and produces a set of results or matches for the query. Applying the concept of pattern matching, we can create math formula query languages, which take advantage of the use of wildcards to allow for more expressiveness in the creation of queries. For example, the wildcard # (number sign) could represent a matrix. Unfortunately, integration of formula query languages and pen-based interfaces is limited. Therefore, we propose a novel query language suitable for pen-based interfaces and implement a proof of concept using MathBrush, an existing pen-based system. We propose multiple interfaces that make use of the language. Lastly, we present an experimental framework for the interfaces.

Additional Key Words and Phrases

Math retrieval system, math search, pen-based devices, touch-based devices, wildcard matching, formula query language, pattern matching

1 INTRODUCTION

Mathematical expressions and formulae are ubiquitous in scientific literature. The use of such expressions is required to describe problems and theories using a universally accepted formal language. With the ever-growing availability of mathematical content on the web, the problem of math search increases its importance and difficulty. Conventional search engines are tailored for natural language textual data, and are unable to properly handle math search [70]. More specifically, (1) inputting and (2) resolving a math query poses multiple challenges. For the former part, it becomes a cumbersome problem for the user figuring out how to enter mathematical expressions. For the latter part, traditional keyword-based query techniques (e.g. exact matching, partial matching) might not yield satisfactory results.

We can extend the scope for the first problem, and consider the task of inputting mathematical expressions into technological devices. The two most common methods include typesetting languages, such as LaTeX, and GUI selection menus, such as the one present in the Microsoft Office Suite [29, 30]. However, the former requires the user to interact with a plethora of codes and synthetic rules, whereas the latter requires the user to interact with an extensive set of menu options and math symbols, using mouse and keyboard as input devices.

Another option, and also a more natural approach, is the ability to input handwritten mathematical expressions using pen-based or touch-based devices, mimicking the action of writing them on a piece of paper. This field is also enhanced by the recent advances in mobile and touch screen devices.

Handwritten math recognition has been proposed for various scenarios:

- Exporting the expression for display or printing. The user may want the LaTeX string representation or simply an image to copy and paste in a word processing application.

- Evaluating or simplifying the expression. There are many software libraries known as Computer Algebra Systems (CASs) used for this task. Some examples are Maple and Mathematica.
- Math query. In this case, the recognized expression is used as a query to search for documents with similar math notation.
- Teaching and tutoring sessions. Some math handwriting systems are used to assist instructors in the classroom setting, targeting students as their primary users.

Handwritten recognition can be divided into online and offline. In the former case, handwriting is captured as a series of strokes, trajectories of pen/hand tip movements, entered by the user. The latter group analyses and classifies digital images into different classes [2, 87]. In this work we focus on online recognition systems, because they are more suitable to work in conjunction with math retrieval systems, and they also have been widely used for pen/touch input devices.

Extensive work has been done to tackle the task of handwritten math recognition. However the majority of the approaches decouple the user interaction and the input interface from the recognition process itself and just solve the latter. In particular, many neural network based approaches rely on the use of deep learning architectures (CNNs, DNNs, RNNs, LSTM-RNNs) to achieve a good performance [58]. Such approaches are not concerned with how the handwritten math expressions are captured. Instead, they use publicly available mathematical symbol datasets. We classify these approaches as offline recognition, as they basically classify an input dataset of digital images that correspond to math notations. On the other hand, online mathematical recognition systems are more scarce, but there are several in the field. Online handwritten math recognition systems are normally developed to be used in conjunction with pen-based devices and have a visual interface that contains a set of options and menus that allow the user to interact with the system, for example by adding new mathematical expressions, editing them, or exporting them for further interaction with other software applications such as CASs.

Besides recognition, math retrieval systems are also concerned with the information retrieval process. Users start with a particular information need, which is expressed as a query. The system then matches the searchable documents with the query, in order to produce a set of relevant results. Techniques such as relevance feedback and relevance ranking can also be used to enhance the effectiveness of the retrieval process [74]. When math retrieval systems support handwritten math recognition, the recognized expression acts as the query.

In the field of pattern matching the use of wildcards allows users to obtain approximate matches given a particular query. Wildcards are a common way of describing unknown data. In the context of natural language, a wildcard can be simply defined as a single character that matches any symbol in the input alphabet [14, 53]. However, this definition can become more complex. With regular expression (regexp) matching users have at their disposal an extensive set of wildcards that can be combined to create expressive and yet succinct queries. For example, in Javascript, the regexp $[1-9]$ will match any digit from 1 to 9, and the regexp $[a-zA-Z]$ will match any letter from the English alphabet. As we can observe, in a natural language context, wildcards provide a lot of expressiveness when formulating queries, and also provide a concise method to capture user's queries [49, 65]. Conversely, the majority of math recognition systems are unable to deal with wildcard symbols.

This work is motivated by the lack of expressiveness and flexibility in current pen-based math query interfaces. Wildcard support in handwritten math query systems is limited. Moreover, the works that propose a math query language with extensive wildcard support are not really concerned with the user interaction aspect of the system. For example, some of them contain redundant or ambiguous rules in the language that could affect the input of math expressions using pen-based/touch-based interfaces. This paper analyses multiple aspects that researchers should consider when developing pen-based math query interfaces. Moreover, we provide multiple guidelines that could be helpful when developing such interfaces. At a high-level the aspects we consider can be divided in these groups:

- *Gesture support and button menus.* Many math query interfaces use a set of toolbar menus to display the available actions [20]. However, traditional toolbar buttons can distract the user from what she is doing. They also can be perceived as time-consuming as the user has to change her focus between the toolbar and the editing area. Lastly, they can be difficult to properly tap/toggle with a pen or the hand. We present existing techniques that rely on the use of gestures and floating button menus to allow fluidity in the interaction and allow for an easier manipulation of the handwritten math expressions [19, 34, 51, 80]. The intention with both is to eliminate as much interference as possible, reducing time consumed and user frustration. For example, a common gesture is the scratch-out gesture, which allows a user to delete or remove the part that was scratched from the editing area.
- *Feedback mechanisms.* Query interfaces can make use of feedback mechanisms to communicate a particular message to the user. This message can be the termination of an operation, the emphasis of particular sections in the input, the explanation of a menu option, the selection or activation of a particular operation, and many more. Visual feedback is by far the most common, although in recent years tactile and auditory feedback have gained popularity. Different types of feedback mechanisms can be used in conjunction, for example using auditory feedback to complement visual feedback. One of the intentions behind feedback mechanisms is to make menu selection more efficient and intuitive for the user [61].
- *Math formula query language.* The concept of wildcard matching has been applied to math search, in order to create math query languages [3, 26, 31]. Math query languages are inspired by traditional pattern and regular expression matching. They emulate the use of wildcards to allow for approximate matchings, increasing the expressiveness and reducing the time it takes to construct particular queries. Approximate matching allows users to find similar or related expressions. However, depending how the system treats the relatedness between expressions, for example with a query that is too broad, the result-set could be far from precise.
- *Quality attributes.* The previous aspects relate to how the system captures mathematical input and constructs the query for the information retrieval process. For users to benefit from these features, we need to consider multiple quality attributes. For example, how do users discover the gestures available? how do users specify that a particular symbol is a wildcard? can users modify the action associated with gestures? Multiple experimental studies have analyzed user interaction with pen-based interfaces and provide useful advice on how to develop such systems [5, 41, 57, 82]. We present the insights from these studies.

The rest of this paper is organized as follows. In Section 2, we present background information, review previous literature and compare existing handwritten math systems, and formula query languages. In Section 3, we propose a novel formula query language that draws ideas from existing languages. Then, in Section 4 we explore the integration of our language with an existing pen-based handwritten math system, tackling some of the barriers encountered when capturing user input. Section 5 presents an alternative interface that uses UI components to assist users when entering queries. Next, Section 6 considers exceptional scenarios when processing user input. We propose an experimental framework in Section 7 and conclude the paper in Section 8.

2 RELATED WORK

2.1 Handwritten Mathematical Recognition

Handwritten mathematical recognition can be roughly decoupled into four tasks [74]:

- *Symbol segmentation.* Strokes are grouped based on the symbol they represent. A major consideration in this task is that symbols may be composed of non-consecutive strokes (according to the input order).
- *Symbol recognition.* Each symbol is assigned a class or label. This is also a demanding task because of two problems (1) extensive number of classes and (2) similarity between classes, which causes ambiguity and might yield a wrong result. The latter problem becomes worse if we consider that some mathematical

symbols have more than one written representation. Moreover handwritten math can have significant differences based on the person who wrote it.

- *Structural analysis.* This phase takes care of identifying spatial relations between symbols and producing a mathematical interpretation. Unlike written English, which typically has a horizontal relation (i.e. read from left to right), mathematical expressions have a more diverse set of relations such as vertical and oblique relations such as subscript, superscript, above, below, and containment relationships [74].
- *Mathematical content interpretation.* This phase uses the symbols and their layout to derive a particular configuration of mathematical syntax and semantics. Normally, a syntax tree is generated for the handwritten input. The tree can be used to evaluate the expression or to export it to other applications.

Regarding math retrieval systems, Zanibbi and Blostein [74] state that there are four main challenges :

- *Query formulation.* Systems need to provide effective user interfaces for query formulation. This also requires establishing which types of queries are useful and feasible.
- *Normalization.* Similar to text-based retrieval, math expressions need to be reduced to canonical forms. This prevents *mismatches between equivalent expressions with different representations.*
- *Indexing and matching.* Document representation and the similarity measures used to calculate matches against the math query can have an enormous impact in the retrieval performance.
- *Relevance feedback.* Users can provide such feedback when examining the retrieval results, allowing the system to refine the query.

Existing recognition approaches can be classified into two categories: sequential and integrated [30, 85]. The former treat the aforementioned tasks in a sequential manner. The main drawback is that this technique propagates each error across the tasks, ending with the accumulated errors in the content interpretation phase. The latter group takes into account the natural relationship between the tasks. Additionally, integrated approaches take advantage of contextual information to prevent the generation of invalid expressions. Existing techniques make use of grammar parsing techniques to capture the contextual information. The caveat of this group is that it can be time consuming to manually generate the grammar and computationally expensive to perform the parsing process.

Most of the early-stage works belong to the sequential group [48, 72, 76, 84]. In the second group we can find more recent approaches [4, 7, 12, 29, 46].

For instance, Hu and Zannibbi [27] propose a parsing technique based on visual features. The authors argue that *mathematical expression recognition can be posed as searching for a Symbol Layout Tree (SLT) representing symbols and their associated spatial relationships in a graph of handwritten strokes.* Based on this idea, the authors propose a Maximum Spanning Tree (MST) parser that performs a two-stage process: (1) symbol segmentation, and (2) labeling of spatial relationship between symbols.

Besides the above groups, a third category emerged in recent years. Today, many math recognition techniques rely on the use of neural network architectures [17, 58, 63, 83, 86]. NN-based approaches also consider the relationship between the tasks in order to decrease the number of recognition errors. However, some of these approaches do not require the use of grammars, reducing the computational cost. For example, Zhang proposes the use of a Bidirectional Long Short-term Memory (BLSTM) network, commonly used in text and speech recognition tasks, to capture the contextual information [85]. Most studies in the last category are concerned with the math recognition step but do not consider the user interaction or how the mathematical expressions are captured; instead they rely on available handwritten mathematical datasets.

2.2 CROHME Competition

Started in 2011, CROHME is one of the most well-regarded competitions on Handwritten Mathematical Recognition. CROHME has helped to develop the field of handwritten math recognition providing contributions such as

multiple datasets, competition tasks, evaluation metrics, and participating systems. One of the insights proposed by Mouchère et al [55], derived from analyzing the results of past CROHME competitions, is that handwritten math recognition system could be enhanced by using a statistical language model, such as n-gram or skip-gram. These models have been successfully implemented to improve natural language recognition.

2.3 Math Retrieval Systems

The **Tangent** math search engine was originally published by Stalnaker [15]. Tangent allows the indexing and retrieval of math expressions using the concept of *inverted index* on symbol pairs from layout trees. The authors initially conducted two experimental studies of Tangent: a usability study where participants scored the retrieval results and a performance analysis [69]. The authors used two datasets in the experiments: Math Retrieval Collection (MREC), a collection of 300.000+ academic publications; and the Wikipedia corpus. The authors compared Tangent against Apache Lucene [36] in the experiments, and found that Tangent was able to achieve similar metrics (even surpassing it in some cases) to Lucene, a much mature project. More recent works have carried out additional experiments with Tangent [62, 79], and explored the use of new similarity metrics to generate the result ranking [77, 78].

The **NIST DLMF** (Digital Library of Mathematical Functions) project was started in 1996 by the National Institute of Standards and Technology (NIST) [44, 52] [1]. The project can be considered as a digital version of the *Handbook of mathematical functions: with formulas, graphs, and mathematical tables* [1]. DLMF provides extensive search capabilities. To support mathematical search, DLMF extends conventional text-based search by introducing additional elements in the engine: preprocessing of math expressions into textual form, definition of a math query language to express formula queries, and transformation of search queries into special forms. DLMF supports two wildcards, $, which stands for zero or more alphanumeric characters, and ?, which stands for zero or one alphanumeric character.

MathWebSearch is a search engine that uses a semantic approach to address math queries, finding formulae by their structure instead of their presentation [33]. MathWebSearch provides template palettes for standard mathematical structures such as fractions and integrals. By simply clicking on any template a textual representation is added in the query. Users can also type the mathematical input (they need to be familiar with the textual representation).

MIaS (Math Indexer and Searcher) is a math-aware full-text based search engine [68]. MIaS web interface is called **WebMIaS** [43], and accepts math queries in TeX or MathML notation combined with text queries. TeX queries are converted on-the-fly into MathML tree representation.

EgoMath2 is another full text search engine based on Egothor [2] [54]. The authors assessed EgoMath2's retrieval performance with data from Wikipedia (dump of English articles from January 2011). **WikiMirs** is a system proposed to facilitate mathematical formula retrieval in Wikipedia [28]. WikiMirs proposes a *presentation tree parser to parse formula structures from a layout presentation markup*. Additionaly, WikiMirs uses a hierarchical generalization technique to generate fine-grained sub-trees, which represent index terms.

Approach0 is a search engine aimed to provide better search experience for mathematical QA websites [88]. Currently, Approach0 can crawl information from Math StackExchange (QA website) [3]. Approach0 supports the wildcard ?, which represents any math expression except a single symbol.

[1] https://dlmf.nist.gov/
[2] http://www.egothor.org
[3] https://math.stackexchange.com/

Other math search engines have been proposed from academic research labs and open source projects [42, 56, 59], commercial solutions [71], publicly available websites (Springer LaTeXSearch [4], Wolfram|Alpha [5]), and government initiatives [6]. Most of these works correspond to text based search engines.

2.4 Online Handwritten Mathematical Recognition Systems

MoboMath is a handwriting recognition solution developed by Enventra (currently discontinued) [19]. MoboMath allows users to export the recognized output to other documents and applications, using standard formats such as MathML, LaTeX, or JPEG image format. MoboMath support four basic commands (Enter, Undo, Redo, Clear) and four types of mathematical operations based on gestures: Text selection (tapping and circle gesture), Deletion (x-out gesture, and scratch out gesture), Character/Symbol alternation (i.e. we can tap the incorrect character repeatedly to replace it with alternate recognition results), Addition of ink (space gesture to add extra blank space), Moving and copying (drag-and-drop and use of short up/down strokes to move subexpressions between subscript, factor, and exponent).

MathBrush is a pen-based mathematical system designed for tablets PCs and iOS devices [34, 35] (a prototype web version is also available upon request from the authors [21]). MathBrush combines handwritten math recognition and integration with multiple CASs (e.g. Mathematica, Maple, and Sage). MathBrush is able to recognize an extensive set of expressions such as integrals, matrices, and polynomials. MathBrush offers nine basic context menus present in the application toolbar: Math operations menu, Sharing menu, Export menu, Help menu, Settings menu, Account Menu, Save, Delete/Clear, and Undo. MathBrush has limited support for gestures, including the scratch out gesture (which deletes input characters) and tap/select gesture (which can move expressions or display context menus).

MathPad^2 is a pen-based tablet PC application designed to create dynamic illustrations used for exploring mathematics and physics concepts [37, 38]. MathPad^2 uses a modeless gestural interface, with the idea that gestures should not interfere with the entry of drawings but still be natural enough to feel fluid. Three key features of MathPad^2 are (1) context sensitivity to determine what operations to perform with a single gesture, (2) notion of punctuated gestures, compound gestures with one or more strokes and terminal punctuation, that help to distinguish gestures from mathematics and drawings, and (3) one-to-many pairing between gestures and operations.

Math*S*pad Tablet (MST) is a structure editor that aims to facilitate the presentation and manipulation of handwritten mathematical expressions [50, 51]. MST provides an extensive set of gestures through the *ApplicationGestures* library made available through the Tablet PC API. Examples of gestures include the circle gesture (used to select the content inside the circle), right-down and right-up gestures (used to add and remove vertical space), and check gesture ✓ (used to trigger the copy action). Gestures are also used to apply mathematical rules (e.g. distributivity, symmetry). The *MST library* allows the association between gestures and actions. The library allows users to edit and adapt the gestures according to their needs.

Freehand Formula Entry System (FFES) is a pen-based equation editor. FFES allows the freehand entry and editing of formulae using pen and tablet [67]. FFES has four main modules: a (1) symbol recognizer, which (for each stroke) returns a list of potential characters that the stroke may represent, each option has associated a confidence level; a (2) stroke grouper that segments the input strokes into individual characters; a (3) formula processor that generates the math expression for a set of symbols; and a (4) user interface that allows for easy entry, manipulation and correction of formulae. Upon recognition of the mathematical input, FFES annotates the symbols (using small typeface labels) and offers gesture-based correction of some recognition errors. FFES's

[4] https://link.springer.com/
[5] https://www.wolframalpha.com/

character recognizer and expression partitioner are based on the Caltech Interface Tools (CIT) library [66], while the parser is a command line tool called DRACULAE [75, 76].

MathPaper is a system for fluid pen-based entry and editing of mathematics with support for interactive computation [80]. MathPaper supports extended notations for algorithm sketching through *AlgoSketch* [40]. MathPaper supports multiple types of gestures: computation gestures ($\longrightarrow, \Longrightarrow$), used to evaluate expressions; graphing gestures ($\longleftarrow, \longrightarrow, \uparrow, \downarrow, \nearrow, \searrow, \swarrow, \nwarrow$), used to graph expressions; space management gestures ($|, \neg, -$), used to add horizontal or vertical spaces in matrices; and other gestures such as circle gesture (to select subexpressions), and drag-out and scratch-out gestures (to delete subexpressions). Additionally, MathPaper uses a technique inspired by *Shadow Buttons* and *Hover Widgets* [25, 47] in order to associate more commands to menu buttons. MathPaper draws a green square under the ink on the display. This square is hidden during normal pen use, but hovering it causes a cluster of buttons to appear. If the pen moves towards the buttons, these can be interacted with as usual, otherwise the buttons simply disappear. To help users discover the available gestures and features, MathPaper displays a preview animation to the user which shows the corresponding command when the user performs an action using a command. *For instance, if the user selects Delete from the menu, the system draws a scribble as an animation over the ink before deletion.*

2.5 Gesture Support and Button Menus

Traditionally, user interfaces present available actions to the user using toolbars and static menus. Moreover, software applications such as the Microsoft Office Suite provide additional functionality with each new version. The main caveat is that such interfaces end up cluttered with a myriad of options and menus, from which users only interact with a small fraction [20]. Fortunately, in comparison with highly configurable applications, such as word processors or spreadsheets, the majority of pen-based math query interfaces have a narrower set of actions and core features. Previously, we presented some online handwritten math recognition systems. The main approaches used to display the actions in the interfaces are static command menus, gestural menus, and hidden menus. For static command menus, MoboMath and MathBrush can serve as examples. For gestural menus, MathPaper and MST support the most extensive set of gestures. For hidden menus, the best example is MathPaper, where a hidden menu is placed outside of the drawing area, but the user can interact with the menu by simply hovering over it.

Interestingly, most of the systems opt to support only a few actions, whether it is in the form of commands, gestures or extra button menus. Some systems offer an extensive number of gestures and allow users to personalize them (associate particular action with the available gestures). This is likely due to the fact that gestures normally do not occupy visual space in the interface. The caveat of increasing the number of available gestures is that users need to learn how to use them (we address this in a later section). Moreover, MathPaper combines the use of gestures and hidden menus in order to maintain a reduced set of gestures, facilitating the user interaction.

A common challenge for user interfaces is teaching users which features are available and how they can interact with them. During the interaction users engage in a learning process that allows them to discover the most useful features (for them), depending on the tasks they want to perform. Some researchers refer to this process as learnability. They have analyzed how learnability varies based on multiple factors.

Anderson and Bischof [5] experimented with multiple gesture menus and concluded that there is an inverse relationship between ease of use and learning (retention and transfer). In other words, when the interfaces are too simple and easy to use, users do not take care to learn how the system works, as they can figure it out every time they interact with the system. Negulescu and Ruiz [57] found that creating an explicit mode for non-command interaction was a preferred alternative by users to communicate the features and gestures available in the system.

Bragdon et al. [10] proposed a gesture learning systems called GestureBar. GestureBar uses a simple toolbar that acts as a sandbox environment to explore the effect of gestures. Another system called OctoPocus, proposed

by Bau and Mackay [8], presents a learning interface for single-stroke gestures. When users initiate a gesture, OctoPocus displays an animation preview of all the potential gestures that can be performed.

Another attribute of interest when considering gesture menus is customization. We can observe that many existing pen-based math interfaces allow users to personalize the actions associated with gestures, sometimes they even allow modifying the gesture itself or to introduce new gestures [41]. However, it is important to make sure that gestures still conserve some basic guidelines; such as being simple, distinct and self-revealing (intuitive or guessable) [82]. Systems with support for gestures should strive to reach a balance between these two aspects.

2.6 Feedback Mechanisms

User interfaces can rely on different types of feedback to communicate particular messages to the user. For instance, MathPaper makes use of animations to explain how gestures works. Visual feedback often is used in user interfaces, and can be integrated in a myriad of forms such as static documentation and help menus, tooltips, animations to explain new features (commonly used on games), and action previews [16].

Ciampa [13] examined multiple types of online password feedback mechanisms (displayed when users are setting up a new password). The experimental study indicated that in all cases *the feedback mechanisms significantly influenced users with lower password ratings to choose a more secure password.* In other words, users were receptive to the feedback mechanisms and updated their input accordingly.

LaViola et al. [39] explored multiple techniques for visualizing the machine interpretation of handwritten mathematics such as replacing user handwritten math with typeset math (replacing user's ink with a predefined font), coloring, and changing the size (with 2 variants: small and large size) of the handwritten ink. The authors found that people tend to prefer the coloring and small font size techniques.

Tactile and auditory feedback can be used to complement visual feedback. For example, systems can provide simple auditory feedback during the transition between menu items, or more complex continuous feedback at specific moments during the interaction [61].

Regarding the use of feedback mechanisms on pen-based math systems, customization becomes an attribute of interest. As with gesture menus, the system could allow users to personalize the feedback mechanisms, to choose the ones they prefer, or even to disable them because they are considered intrusive. However, we need to be careful not to affect the usability of the system, (e.g. some feedback mechanisms can be crucial to offer a smooth interaction).

2.7 Math Formula Query Languages

Most pen-based interfaces provide either no support or only a limited support for wildcards. Some of the systems mentioned in previous sections provide support for a few wildcards: ? and $*$. The former represents exactly one arbitrary character in the math expression. The latter stands for zero or more arbitrary characters in the expression. On the other hand, some researchers have proposed math formula query languages that exploit the use of wildcards to allow powerful and expressive queries. Below we present these works and highlight their advantages and drawbacks.

2.7.1 Altamimi and Youssef

Altamimi and Youssef [3] present five sets of symbols that capture user needs in the area of math search: (1) separator symbols, (2) character-level wildcards (within a term), (3) component-level wild-cards (within a list), (4) term-level wildcards (within an expression), and (5) bounded wildcards.

The separator symbols are @ (at sign), ; (semicolon), and , (comma). @ indicates a function, ; separates the rows in a matrix, and , separates entries in a sequence of elements (e.g. arguments in a function, columns in a matrix).

Character-level wildcards correspond to ? and $. ? stands for a single character, whereas $ stands for multiple characters.

Component-level wildcards represent a sequence of components within a list/sequence. The wildcards are −− (two dashes) and ... (three dots). −− stands for a single component, whereas ... stands for multiple components.

Term-level wildcards represent a sequence of terms within an expression. The wildcards are $ and ... (three dots). $ stands for a single term, whereas ... stands for multiple terms.

Bounded wildcards have multiple levels of abstraction.

- First level. The $ symbol used alone represents any arbitrary term. For example, the query $\$^2 + \$^2 = 20$ matches both $x^2 + y^2 = 20$ and $2^2 + 4^2 = 20$.
- Second level. When the $ symbol is followed by a number, the number designates an identifier. For example $\cos^2\$1 + \sin^2\$2$ matches $\cos^2 x + \sin^2 y$.
- Third level. The query can specify whether the term is a variable $v or a number $n. For example, the query $n is matched by 2 but is not matched by x.
- Data types. Users can specify the data type of the term. The options are: real $R, integer $Z, rational $Q, complex $C, polar $P, and function $F.

The wildcard sets are limited and redundant. Except for the bounded wildcards, the use of wildcards is context-dependant: $ can represent multiple characters if treated as a character-level wildcard; but it can also represent a single term, if treated as a component-level wildcard. Similarly ... can represent multiple components within a list, or multiple terms within a expression. This problem can lead to ambiguity. On one hand, the system would have to consider the context in which a wildcard is used when interpreting its meaning. On the other hand, if users have to limit themselves to a particular level when constructing queries (character, component, or term), this limitation would be seen as a barrier and represent additional effort from the users (e.g. more time spent constructing the queries). As for the bounded wildcards, they seem to allow the interaction between different levels of abstraction, but still limited, especially considering that the language does not allow sub-query matching.

2.7.2 *Kamali and Tompa*

Kamali and Tompa [31] propose a query language that allows approximate matching of math expressions. The authors define four wildcards: [Ni] which matches any number, [Vi] which matches any variable, [Oi] which matches any operator, [Ei] which matches any expression. i is an optional value, and corresponds to a natural number that designates the index (identifier) of a wildcard (i.e. same identifier means same expression on the query). Wildcards can impose constraints by using the where clause:

- Number wildcards. Users can constrain the range, domain or data type (e.g. natural, real). For example the query $x^{[N1]}$ where $1 \leq N1 \leq 5$ matches x^2 but not x^{-1}.
- Operator and variable wildcards. Users can constrain the set of possibilities. For variables, it corresponds to the set of possible names, e.g. [V] where $V \in x, y, z$. For operators, it corresponds to the actual operators, e.g. [O] where $O \in \pm, \times$.
- Expression wildcards. Users can define that the expression must contain particular subexpressions, e.g. [E] where E contains Q′, Q′ can be any query, even one with wildcards and constraints. For example, the query [E1] + 1 where E1 contains x^2 matches $x^2 + 1$ and $\sqrt{x^2 + y + 1}$ but not $x + 1$.
- Optional patterns are supported using braces {}. For example, $x^2\{+[N]\}$ matches both $x^2 + 1$ and x^2.

In comparison with the previous work, this query language allows for more powerful and expressive math queries. The possibility for users to add constraints to the wildcards is intuitive and simple to grasp, and resembles *advanced search menus* of conventional search engines where users can add new constraints or choose between a set of predefined constraints. Moreover, the wildcard set is fairly simple and easy to understand. Sub-expression matching is one of the main contributions of this language, the possibility to add nested constraints when using

the expression wildcard combines the benefits of approximate matching and constraint-based search, increasing the number of results but still precise enough to discard unrelated entries based on the user needs.

2.7.3 *Guo et al.*

MathSearch is a formula-based information retrieval system developed and maintained by Lanzhou University. One of the components of MathSearch is the query module designed to capture and process user queries. For this task, Guo et al. [26] propose MQL, a mathematical formula query language. MQL contains two forms: MQLS and MQLX. MQLS is a character-string form and the human readable version of MQL. MQLS is intended to help users input the math expression or query they want to search. MQLX is the machine-oriented version of MQL. MQLX relies on MathML and XML elements to describe the retrieval content. In this paper, we focus on MQLS, as we are mostly concerned with how users construct queries.

MQL supports six wildcard levels. For each wildcard, MQL defines a set of constraint attributes that can be used to further reduce the search result.

- Operand level. The symbol `?` stands for an arbitrary single operand. MQL recognizes four types of operands: number, symbols, constants, and variables. The `class` attribute is a natural number that denotes the identifier of the wildcard (i.e. same-type wildcards with the same id must correspond to the same expression in the result entries). The `kind` attribute denotes whether the operand is `n` (number) , `v` (variable), or `c` (constant). The `type` attribute denotes the data type and it can be assigned one of these values: `R` (real) , `Z` (integer), `Q` (rational), `C` (complex). The `value` attribute restricts the set of possible values for the operand (it corresponds to a list of comma separated values). Users can also restrict the operand to a value range using the `min` and `max` attributes. For instance, `?[value = 3, 0.01, x] + y` matches `3 + y`, `x + y` and `0.01 + y`; while `?[min = 10 max = 100] + x^2` matches `22.2 + x^2`, and `98.4 + x^2`.
- Expression level. The symbol `$` stands for an arbitrary single sub-expression. This wildcard has the attributes `class`, `value`, `min` and `max`. It introduces two new attributes: `num` and `pattern`, which denotes a formula pattern. `num`, `min` and `max` restrict the number of operands in the expression. Both the `value` and `pattern` attributes can contain nested wildcard queries (similar to the expression wildcard in the previous work). For example, `$[value = $/$]` matches `x/y` and `(a + 2)/b`; while `$[pattern = $/$]` matches `x + (a + 2)/b`, `x/y` and `(a + 2)/b`.
- Sequence level. The symbol `--` stands for an arbitrary single sequence (list of comma-separated values). This wildcard has the attributes `num`, `min` (which denotes the minimum number of items in the sequence) and `max` (which denotes the maximum number of items in the sequence).
- Operator level. The symbol `~` stands for an arbitrary single mathematical operator. The only attribute for this wildcard is `priority`, which restricts the operator to a particular priority. There are four priority values: 1, which represents `>`, `<`, and `=` (greater than, less than, and equal to); 2, which represents `+`, and `-` (addition, and subtraction); 3, which represents `/`, `*`, and `%` (division, multiplication, and modulo); and 4, which represents `^` (powers). Users can choose among multiple operations when restricting the priority: `>` (greater than), `<` (less than), `=` (equal to), `!=` (different from), `>=` (greater or equal than), and `<=` (less or equal than). For instance, `x ~ [priority >= 2] z` matches `x + z`, `x * z` and `x ^ z` but it does not match `x = z` or `x >= z`
- Function level. The symbol `@` stands for an arbitrary single function. The attributes `num`, `min` and `max` restrict the number of operands in the expression. The wildcard also supports the attributes `value`, `pvalue`, `pattern`, and `type`. `value` and `pattern` restrict the value of the function, whereas `pvalue` restricts the parameters of the function. For example, `@[pvalue = (x, y)]` matches `f(x, y)` and `h(x, y)`; `@[value = x + y]` matches `f(x, y) = x + y` and `g(x, y) = x + y`; and `@[pattern = x + y]` matches `f(x, y) = x + y` and `f(x, y) = x + y + 2`. For `type`, MQL supports a comprehensive set of functions such as `mean`, `sin`, and `cos`. Users can also search for piece-wise functions by inputting a list of semicolon-separated segment-condition

pairs (where ; separates the sub-functions). For instance, @[x − 1, x > 1; 2x, x = 1; x + 1, x < 1] matches

$$f(x) = \begin{cases} x - 1 & x > 1 \\ 2x & x = 1 \\ x + 1 & x < 1 \end{cases}$$

- Matrix level. The symbol # stands for an arbitrary single matrix. This wildcard also has the class attribute. The kind attribute can take one of these values: v (the matrix is composed of variables), n (the matrix is composed of numbers), or m (the matrix is composed of a mixture of variables and numbers). The type attribute can take one of these values: I (identity matrix), Z (zero matrix), or D (diagonal matrix). nrow and ncolumn restrict the number of rows and columns respectively. For example, #[nrow = 3 ncolumn = 3 type = D] matches both matrices, while #[nrow = 3 ncolumn = 3 type = I] matches the first matrix but not the second one (any identity matrix is also a diagonal matrix).

$$\begin{bmatrix} x^2 & 0 & 0 \\ 0 & 45 & 0 \\ 0 & 0 & \pi + 2\mu \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

MQL also supports combination queries (multiple formula queries combined) but in a limited fashion, the language relies on commonly-used Boolean operators && (AND)- || (OR), and ! (NOT), to treat combination queries as Boolean expressions.

Compared with the previous papers, MQL provides a more formal definition of the query formula language, However the language is not designed for pen-based interfaces, as the authors assume that users have access to a keyboard as input device. In a pen-based interface, the excessive amount of wildcard parameters (or attributes) reduces the simplicity and ease of use of the language. Many of the wildcard attributes are redundant or lack clarity. As we saw with some of the examples, different wildcard attributes can be used to represent the same constraint. Additionally, the authors do not provide enough detail on how multiple wildcard attributes can be used at once. For example, for the function wildcard, is it required for all the variables in the value of the function to appear in the parameters list?; or for the operand wildcard, should the type attribute be disabled when the user sets the attribute kind to v?. Another shortcoming is the lack of explanation and examples for nesting wildcards and queries when using the attributes value and pattern.

2.7.4 Quality Attributes

The query languages above disregard how the user interaction would work, especially if we want to use a pen-based input device. Concerns such as how the users indicate the wildcards in the math expressions? Do the users need to write all the wildcard parameters beforehand or they can edit them afterwards? Should the system present wildcard symbols differently from other handwritten symbols? and if so, how?. We can identify multiple quality attributes of interest:

- Discoverability and Learnability. How can users discover the wildcards and learn how they can be used. In the aforementioned query languages each wildcard corresponds to a particular symbol, so a simple idea would be to always recognize such a symbol as a wildcard. However, the limitation of this approach is that users would not be able to use these symbols when inputting mathematical expressions. Moreover, if the recognizer does not identify the wildcard symbols then they would be treated as normal symbols. Another approach would be to indicate which symbols are the wildcards using gestures (e.g. circling, underlining). Again, the limitation here relates to the precision of the recognizer, which could miss some of

the gestures or could misplace the actual wildcard symbols. Lastly, instead of gestures the system could use extra menus (e.g. hidden menus, buttons, commands) for the wildcards, which could reduce potential recognition mistakes.

- Constraint definition. Users can define multiple wildcard constraints which can help to remove irrelevant entries from the result-set. This is a key component that enhances the expressiveness of the wildcards. Therefore, it becomes crucial to properly capture these constraints. A simple idea would be to require users to write all the parameters beforehand (as is proposed then in MQL). However if we consider that the precision of the recognition might not yield the desired input, this approach becomes less viable. Another idea is to allow the addition of the constraints after the user has written the expression. This could be optional, since the user might not want to add constraints at all for some wildcards. The expression wildcard is a special case. As we saw in two of the query languages [26, 31] users can define embedded wildcard queries as constraints for this wildcard. In such cases, the system needs to capture extra handwritten input. This definition of embedded queries can be seen as a wildcard tree, where the wildcard of original query are in the root and (only) the expression wildcards can have descendants. This particular case with the expression wildcard reinforces the idea that an approach where users can add or edit wildcard constraints for the query after inputting it is better suited for pen-math handwritten interfaces.
- Wildcard display. Based on the insights from previous sections we recommend the use of visual feedback to differentiate the wildcard symbols in the mathematical expressions. Previous experimental studies found that *coloring* and *typefacing* are two common visual feedback techniques preferred by users.

3 A FORMULA QUERY LANGUAGE

The query languages aforementioned present an interesting trade-off. Kamali and Tompa offer a more compact and simple language with the caveat that the definition of wildcard constraints lacks formality as is explained mostly through examples. Also, compared to MQL, it does not provide wildcards for sequences, functions, and matrices. Guo et al. propose a more formal definition for the wildcard constraints, introducing the idea of wildcards attributes. The extensive number of available attributes might seem helpful, but on a pen-based interface, it complicates the task of inputting formula queries. Moreover, a more extensive wildcard language can have an unfavorable effect on the recognition accuracy. To account for these shortcomings, we propose a novel formula query language based on MQLS (the character-string form of MQL). Table 1 presents the definition of the wildcards for the language.

- `N` matches any number such as `5`, $-$ `0.25`, and `6.23`. The user can constrain the number to a particular range using the `min_val` (minimum value) and `max_val` (maximum value) constraints. If the user wants to match particular values she can use the `exp_vals` (explicit values) constraint, which corresponds to a list of numbers. The set of potential values that the wildcard can match corresponds to the union between the constrained range (determined by `min_val` and `max_val`) and the set of explicit values.
- `V` matches any variable such as `x`, `A`, and α. The user can constrain the variable name to a custom set of values using the `exp_vals` constraint, which corresponds to a list of variable names.
- `O` matches any operator such as $\times$, $\div$, and $+$. The user can constrain the operator to a custom set of values using the `exp_vals` constraint, which corresponds to a list of operation identifiers. The identifiers are strings that represent mathematical operations.
- `S` matches a sequence of elements (defined as a comma-separated list of elements). The user can constrain the number of items in the sequence to a particular range using the `min_items` (minimum number of items) and `max_items` (maximum number of items) constraints. The constraint `pos_pat` (positional pattern) allows to specify a pattern (nested formula query) for at least one of the items in the function. `pos_pat` requires a tuple of the form $<$ `item_index`, `expression` $>$ where `item_index` corresponds to the 1-indexed position

of the item, and `expression` corresponds to the nested query. `item_index` has some special values: (1) *all* (i.e. the pattern applies to all the items in the sequence); (2) *at least one* (i.e. the pattern applies to at least one of the items in the sequence); (3) negative indexing (i.e. starting from the last element of the sequence), such that for a sequence with `n` elements, index `-i` corresponds to the position `n-i+1`. In other words `-1` would correspond to the last element in the sequence.

- `F` matches a function such as `f(x)`, `g(x+y,2)`, and `P(1,2)`. The user can specify that the function name belongs to a custom set of values using the `exp_vals` constraint, which corresponds to a list of function names, such as alphabetic letters or well-known functions (e.g. sin, cos, mod). For the function parameter we decided to reuse the other wildcards, and thus, the `param` (function parameter) simply requires a wildcard definition. For example, if we want a function with multiple items we simply need to set the `param` constraint to the `S` wildcard.
- `M` matches a matrix. The user can specify the dimensions of the matrix using the `num_cols` (number of columns) and `num_rows` (number of rows). For commonly-used types of matrices the user can specify the `type` constraint, using multiple options: *diagonal*, *square*, *upper triangular*, and *lower triangular*. Similarly to the sequence wildcard, the user can specify a positional pattern (nested formula query) for at least one of the cells in the matrix using `pos_pat`. `pos_pat` requires a tuple of the form `<row_index,col_index,expression>` where `row_index` and `col_index` correspond to the 1-indexed coordinates of the cell we want to apply the pattern, and `expression` corresponds to the nested query. `row_index` and `col_index` have some special cases: (1) *all* (i.e. the pattern applies to all the rows for row_index, and all the columns for col_index); (2) *at least one* (i.e. the pattern applies to at least one of the rows for row_index, and at least one of the columns for col_index); (3) negative indexing (i.e. starting from the last element of the sequence).
- `E` matches a mathematical expression. The user can use the `pattern` constraint to specify a nested query. By default, the expression wildcard performs an exact match, but the user can change the behavior to perform partial matching instead.

For any of the wildcards, any constraint is optional. Moreover, the wildcards themselves can also be optional. When the user designates a wildcard as optional, the empty argument becomes a valid match for the wildcard. We did not include the definition of identifiers [26, 31] as constraints, but the language could be easily extended to include such feature.

4 LANGUAGE ADAPTATION WITH MATHBRUSH

Besides the definition of a wildcard language this work also aims to explore how we can integrate the language with existing handwritten mathematical recognition systems. For this, we have selected MathBrush. Flood [21] implemented a web version of MathBrush in order to extend its accessibility. We contacted the Symbolic Computation Group (SCG) at the University of Waterloo [6], which currently maintains MathBrush, to obtain the source code of the web version.

MathBrush is an ongoing project constantly updated and improved. On the other hand, the web version is a prototype version that has not undergone improvements since it was implemented by Flood. As such, the web version has some limitations, the more relevant ones to this paper are

- Integration with Computer Algebra Systems (CASs) is disabled. Mobile versions of MathBrush integrate with multiple CASs (Maple, Sage, and Wolfram|Alpha), which allow users to interact with multiple operations after the application has recognized a valid mathematical expression (e.g evaluate, factor, simplify). The web version has a button that pops up a new tab in the Wolfram|Alpha website, and sends the recognized expression in LaTeX format.

[6] https://www.scg.uwaterloo.ca/mathbrush/people/

Table 1. Wildcard Language Definition

Wildcard	Description	Constraints
N	Number	min_val max_val exp_vals: $[n_1, n_2, \ldots, n_n]$
V	Variable	exp_vals: $[v_1, v_2, \ldots, v_n]$
O	Operator	exp_vals: $[o_1, o_2, \ldots, o_n]$
S	Sequence	min_items max_items pos_pat: (index, expression)
F	Function	exp_names: $[f_1, f_2, \ldots, f_n]$ param
M	Matrix	num_cols num_rows type: diagonal, square, upper triangular, or lower triangular pos_pat: (row_index, col_index, expression)
E	Expression	pattern: nested query matching_type

- Gesture support is almost non-existent. The only gesture supported by the web version is the scratch-out gesture. The addition of new gestures is rather involved as it requires manually implementing the recognition of the gesture and then the implementation of the desired action or effect for the gesture.
- Recognition accuracy has lots of room for improvement. On the mobile versions users can train the recognizer by adding or editing handwritten samples for each symbol (e.g numbers, Greek letters, operators, etc.). However, the web version does not possess this feature, and relies on a base set of symbol samples. Moreover, the recognizer tries to make sense of the mathematical input, meaning that it verifies the syntactic correctness of the input and transforms it accordingly. To illustrate let us consider the query $\frac{-^2}{+^2}$. Here, we can observe a fraction where the numerator is a minus sign superscript 2, and the denominator is a plus sign superscript 2. This does not make sense as a mathematical expression. Therefore, MathBrush will instead recognize an expression like $\frac{-2}{t^2}$, where the numerator no longer has a superscript and the plus sign in the denominator is transformed to a *t*. This is by far the major limitation of the web version, as it constrains the types of language we can define for users to enter their queries.

4.1 MathBrush Layout Adaptation

Figure 1 shows the visual interface of MathBrush. We relocated the recognition panel atop the window, thereby increasing the canvas area where the user draws mathematical expressions. Due to the limitations of the web version, the use of an extended set of symbols (similar to MQL) to input the queries is discouraged. Moreover, the use of gestures to define or constrain wildcards is also not viable. To tackle this we propose a simple idea to input the queries. The first line of the handwritten input corresponds to the **main expression**. Any line after the first line, corresponds to **wildcard definition**, or **wildcard constraint**. A wildcard definition designates a variable from the main expression as a wildcard and requires the user to specify the type of wildcard. A wildcard constraint assigns a constraint to a previously defined wildcard.

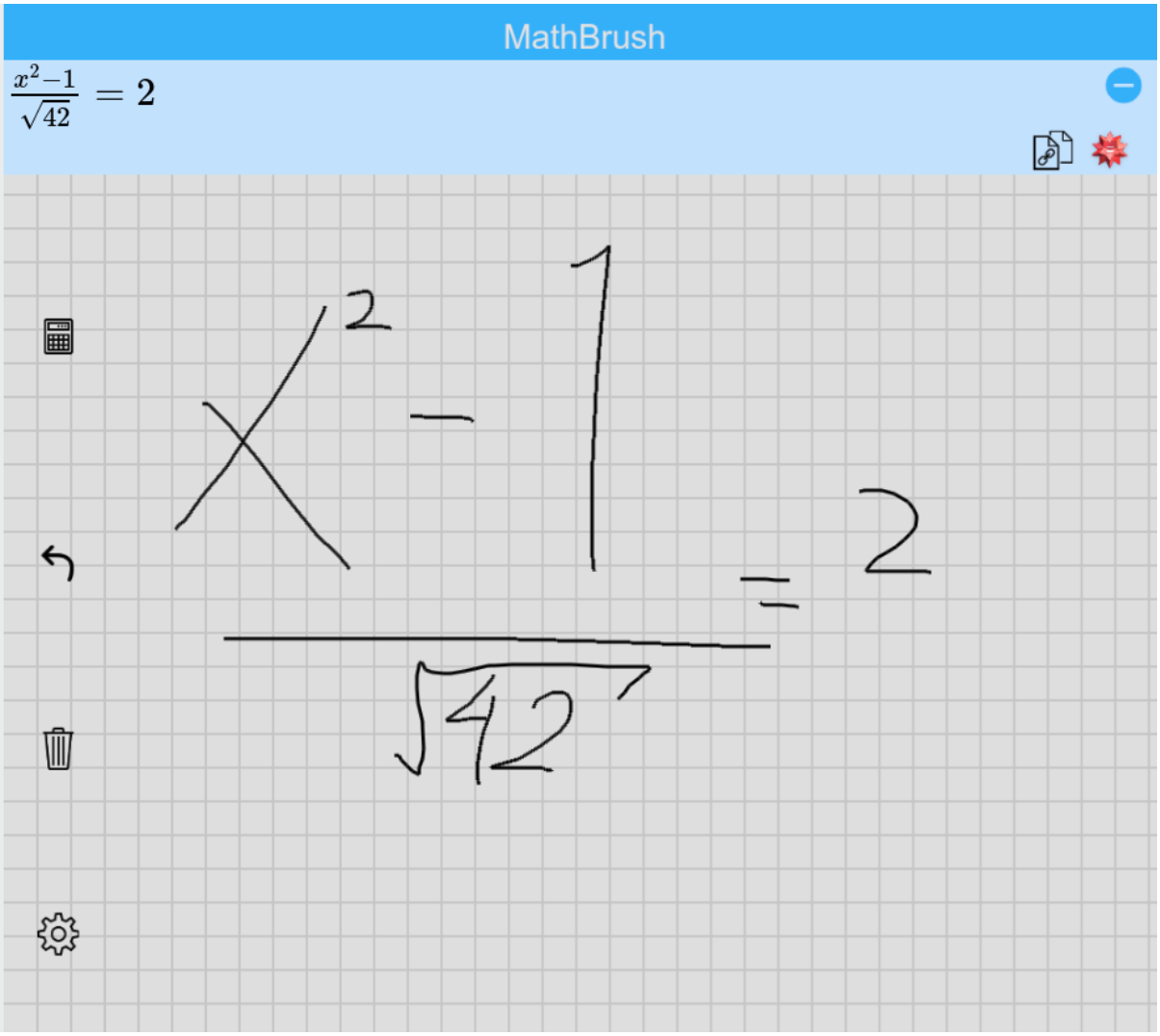


Fig. 1. Interface of MathBrush Web Version

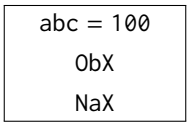
4.2 Wildcard Definition

A wildcard definition corresponds to a string of the form < Tv > where T corresponds to the wildcard type, and v corresponds to the character from the main expression that is being designated as a wildcard. The wildcard type must correspond to one of the wildcard types previously explained: N, V, O, S, F, M, or E. For example, below we can observe two wildcard definitions, x is designated as a numeric wildcard, while y is a variable wildcard.

```
x + y = 2
Nx
Vy
```

The query matches the expressions $10 + x = 2$, $4 + z = 2$, and $-0.29 + y = 2$

To specify that a particular wildcard is optional, users have to add the placeholder symbol X next to the wildcard symbol. For example, This query defines two optional wildcards



This query will match $3 + c = 100$, $10c = 100$, and $-c = 100$

4.3 Wildcard Constraints

For any wildcard constraint, users need to specify which wildcard they are constraining. To represent the wildcard a user just needs to use the character symbol that was previously defined as a wildcard. In other words, users first define the wildcard and then they can use the wildcard variable (which acts as an identifier in the query) to constrain it.

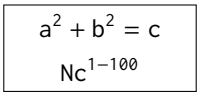
4.3.1 N - Numeric Wildcard

The numeric wildcard can have three types of constraints: min_val, max_val, and exp_vals.

min_val and max_val are specified as a superscript of the wildcard variable. To distinguish between the two, the user needs to use the minus sign symbol − to separate the two parameters (min_val comes first). The constraints are specified in this fashion Nx^{10-100}, which constrains the range of the variable between 10 and 100. The user can omit max_val if she desires, but she must draw a symbol for the min_val. To circumvent a case where the user wants to specify a value for max_val but not for min_val, she must use the placeholder symbol α (lowercase alpha).

For instance, $Nx^{\alpha-100}$ uses the placeholder to avoid defining a value for min_val.

exp_vals requires a list of elements. The user needs to define a list of values separated by the minus sign symbol − enclosed in () (parentheses). Another example:

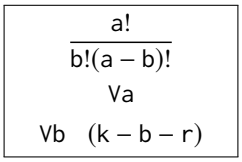


Based on the *Pythagorean theorem*, this query matches would match $a^2 + b^2 = 100$, and $a^2 + b^2 = 25$

4.3.2 V - Variable Wildcard

The variable wildcard has one type of constraint: exp_vals.

As before, exp_vals requires a list of elements. The user needs to define a list of values separated by the minus sign symbol − enclosed in () (parentheses). The difference with the numeric wildcard is that the values corresponds to variable names instead of numbers. For example:



This query will match multiple forms of the *binomial coefficient* formula, such as $\frac{n!}{k!(n-k)!}$ and $\frac{p!}{r!(p-r)!}$, used to compute the number of combinations when repetition is not allowed.

4.3.3 *O - Operator Wildcard*

The operator wildcard has one type of constraint: `exp_vals`.

As before, `exp_vals` requires a list of elements. A peculiarity of this wildcard is that the list of potential values could be treated as finite, as the list of basic math operators is rather small. As we mentioned before, MathBrush tries to make sense of the mathematical correctness of the input, and trying to draw a list of operands (enclosed with parentheses) is a case where the recognizer will transform the input to a mathematically correct one. Lastly, multiple operators can represent the same operation, such as /, ÷ (division); or ∗, x, • (multiplication).

We decided to limit the set of operators that users can input, by assigning alphabetic identifiers to them. Table 2 presents the operators together with their character ID. There are 12 operations, each one with a different ID. In the case of G_e and L_e, we use a subscript *e*. To separate the symbols the users needs to use the separator symbol −. An example:

abc
Ob (G − G_e − M − A)

This query will match a > b, a ≥ b, a × b, and a + b

Table 2. Operator Wildcard. Set of valid operators

Operator	Description	Character ID
>	greater than	G
<	less than	L
=	equals	E
!=, ≠	not equal (different)	N
≥	greater than or equal to	G_e
≤	less than or equal to	L_e
+	addition	A
-	subtraction	S
±	plus-minus sign	PM
/, ÷	division	D
*, x, •	multiplication	M
%	remainder	R

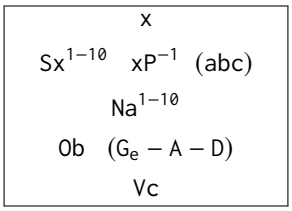
4.3.4 *S - Sequence Wildcard*

The sequence wildcard can have three types of constraints: `min_items`, `max_items`, and `pos_pat`. It is important to note that `min_items` and `max_items` correspond to integers.

Analogous to the numeric wildcard, `min_items` and `max_items` are specified as a superscript of the wildcard variable. To distinguish between the two the user needs to use the minus sign symbol − to separate the two parameters (`min_items` comes first). The constraints are specified in this fashion Sx^{1-10}, which constrains the number of items between 1 and 10. The user can omit either `max_items` or `min_items`.

`pos_pat` requires two elements: an `item_index` and the `expression` to match. To indicate a `pos_pat` constraint, the user must use a placeholder P symbol. `item_index` can be a positive integer (normal indexing), negative integer (negative indexing), 0 (at least one item), or α (all the items). `item_index` is specified as a superscript of the placeholder symbol. Then, the user specifies the `expression` enclosed in () (parentheses).

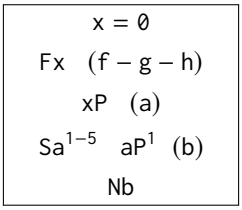
For example, Sx^{-10} xP^{-1} (a + b) represents a sequence with at most 10 elements where the last element corresponds to a + b. Another example:



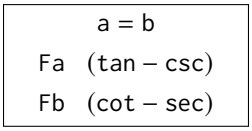
This query will match [1, 2, 3, 1 + x], [10 ÷ z], and [1, 2, 3, 4, 5, 6, 7, 8, 9, 2.22 ≥ p]

4.3.5 *F - Function Wildcard*

The function wildcard can have two types of constraints: `exp_names` and `param`. `exp_names` is analogous to `exp_vals` as it requires a list of elements. The user needs to define a list of function names separated by the minus sign symbol `−`. `param` corresponds to the function parameter and can be associated with other wildcards. To specify `param` the user must add the placeholder symbol `P` and then the nested definition enclosed in `()` (parentheses). It is important to note that when `param` corresponds to a sequence wildcard, during the retrieval phase the query would remove the enclosing symbols from the sequence, since the function wildcard already implies enclosing symbols. Some examples:



The function wildcard matches functions where the function name is either `f`, `g` or `h`. The function must have at least one parameter and at most five parameters, and the first parameter must be a number. This query will match f(0.25, x, p), g(10), and h(−100, α, 2, x + y + z)



This query will match two of the *trigonometric identities* $\tan(\frac{\pi}{2} - \theta) = \cot\theta$ and $\csc(\frac{\pi}{2} - \theta) = \sec\theta$

4.3.6 *M - Matrix Wildcard*

The matrix wildcard can have four types of constraints: `num_cols`, `num_rows`, `type`, and `pos_pat`. `num_cols` and `num_rows` correspond to positive intergers.

Analogous to the sequence wildcard, `num_cols` and `num_rows` are specified as a superscript of the wildcard variable. To distinguish between the two the user needs to use the minus sign symbol `−` to separate the two parameters (`num_rows` comes first). For instance, $\texttt{Mx}^{4-8}$ represents a matrix with 4 rows and 8 columns. The user can omit either `num_cols` or `num_rows`.

`type` allows these options: diagonal matrix, square matrix, upper triangular matrix, and lower triangular matrix. Two types at the same time are allowed as long as they are valid: a square diagonal matrix is valid, whereas an upper triangular and lower triangular matrix does not make sense. To specify `type` the user must use the placeholder symbol `T` superscript the character IDs of the type. The character IDs correspond to the uppercase first character of the type names: `D`, `S`, `U`, and `L`. For instance, $\texttt{MxT}^{\texttt{SD}}$ represents a square diagonal matrix.

pos_pat requires three elements: row_index, col_index and the expression to match. To indicate a pos_pat constraint, the user must use a placeholder P symbol. row_index and col_index can be positive integers (normal indexing), negative integers (negative indexing), 0 (at least one row/column), or α (all the rows/columns). row_index and col_index are specified as a superscript of the placeholder symbol. To separate the two indexes, the user must use the minus sign symbol −. Then, the user specifies the expression enclosed in () (parentheses).

For example, $\mathtt{Mx}^{4-4}$ $\mathtt{xP}^{-1-0}$ $(\mathtt{a}+\mathtt{b})$ represents a square matrix with 16 cells, where at least one of the columns in the last row contains a cell with the value a + b. Additional examples:

$$\boxed{\begin{array}{c}\mathtt{a}+\mathtt{b}\\ \mathtt{Ma}^{4}\quad \mathtt{aT}^{\mathrm{DS}}\quad \mathtt{aP}^{-1--1}\quad (\mu+2\epsilon)\\ \mathtt{Mb}\quad \mathtt{bT}^{\mathrm{U}}\end{array}}$$

This query will match

$$\begin{bmatrix} \mathtt{x}^2 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & \mu+2\epsilon \end{bmatrix} + \begin{bmatrix} 33 & \pi & 44 & 55 \\ 0 & 10 & 4 & 5 \\ 0 & 0 & 22 & 6 \\ 0 & 0 & 0 & 11 \end{bmatrix}$$

$$\boxed{\begin{array}{c}\mathtt{a}=\mathtt{bc}\\ \mathtt{Ma}^{3-1}\\ \mathtt{Mc}^{3-1}\\ \mathtt{Mb}^{3}\quad \mathtt{bT}^{\mathrm{S}}\end{array}}$$

This query matches the *Cabibbo–Kobayashi–Maskawa* (CKM) matrix (also known as quark mixing matrix)

$$\begin{bmatrix} \mathtt{d}' \\ \mathtt{s}' \\ \mathtt{b}' \end{bmatrix} = \begin{bmatrix} \mathtt{V}_{\mathtt{ud}} & \mathtt{V}_{\mathtt{us}} & \mathtt{V}_{\mathtt{ub}} \\ \mathtt{V}_{\mathtt{cd}} & \mathtt{V}_{\mathtt{cs}} & \mathtt{V}_{\mathtt{cb}} \\ \mathtt{V}_{\mathtt{td}} & \mathtt{V}_{\mathtt{ts}} & \mathtt{V}_{\mathtt{tb}} \end{bmatrix} \begin{bmatrix} \mathtt{d} \\ \mathtt{s} \\ \mathtt{b} \end{bmatrix}$$

4.3.7 E - Expression Wildcard

The expression wildcard has two types of constraints: pattern and matching_type.

pattern represents a nested query and can be associated with other wildcards. To specify pattern the user must add the nested definition next to the wildcard enclosed in () (parentheses). By default, this wildcard performs exact matching but the user can switch to partial matching by using the placeholder symbol P. For example, ExP (a + b) will match any expression that includes a + b. Additional examples:

$$\boxed{\begin{array}{c}\mathtt{a}_{\mathtt{n}}=\mathtt{b}\\ \mathtt{EbP}\quad (\mathtt{n}-\mathtt{m})\end{array}}$$

This query matches the general formula to compute the nth term of an arithmetic progression $\mathtt{a}_{\mathtt{n}} = \mathtt{a}_{1} + (\mathtt{n}-\mathtt{m})\mathtt{d}$

$$\boxed{\begin{array}{c}\mathtt{a}+1=0\\ \mathtt{EaP}\quad (\pi)\end{array}}$$

This query matches *Euler's identity* $\mathtt{e}^{\mathtt{i}\pi} + 1 = 0$, often regarded as an example of mathematical beauty.

4.4 Visual Feedback

Herein, we will refer to our first language as *explicit language* and to the second language as *input language*. The input language helps users to input queries on the canvas, given MathBrush limitations; whereas the explicit language facilitates users interpreting the wildcard formula queries they are constructing. We propose integrating both languages in the following way: The input language is used to capture user queries, and the explicit language is used to provide visual feedback of the query.

Let us go back to Figure 1. In MathBrush, the interaction normally starts when the user writes a mathematical expression in the canvas, and then the recognition panel displays the expression. We want to adapt the visual feedback displayed in the recognition panel, using the explicit language.

The explicit language highlights wildcards and their constraints (if any). We use a different color font to represent the wildcard type and also list for each wildcard the corresponding constraints (enclosed in brackets). For each wildcard in the query the explicit language will display an expression with the form $< \mathtt{Tv} > [\mathtt{c}_1, \mathtt{c}_2, ..., \mathtt{c}_n]$ which joins the wildcard definition with the wildcard constraints. We also include the main expression on the feedback, coloring the characters that represent wildcards.

To illustrate this, let us show an example with a query and the resulting feedback.

$$\mathtt{x} = \mathtt{0}$$
$$\mathtt{Fx}\quad (\mathtt{f} - \mathtt{g} - \mathtt{h})$$
$$\mathtt{xP}\quad (\mathtt{a})$$
$$\mathtt{Sa}^{1-5}\quad \mathtt{aP}^{1}\quad (\mathtt{b})$$
$$\mathtt{Nb}$$

For this query, the recognition panel would show the following

$$\mathtt{x} = \mathtt{0}$$
$$\mathtt{x}\ [\mathtt{function},\ \mathtt{exp\_names} = [\mathtt{f}, \mathtt{g}, \mathtt{h}],\ \mathtt{param} = [\mathtt{a}]]$$
$$\mathtt{a}\ [\mathtt{sequence},\ \mathtt{min\_items} = 1,\ \mathtt{max\_items} = 5,\ \mathtt{param} = [\mathtt{b}]]$$
$$\mathtt{b}\ [\mathtt{numeric}]$$

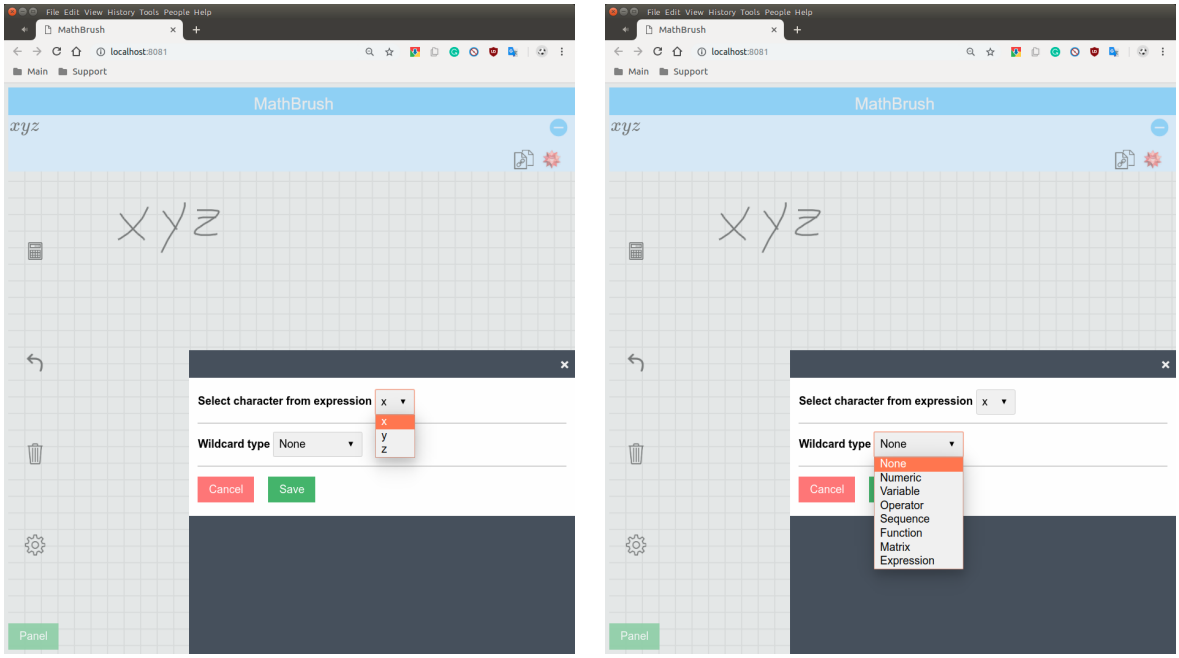
5 AN ALTERNATIVE INTERFACE

The previous section explained how the system uses the existing MathBrush interface (web version) to capture queries. Herein, we refer to this method as the **base interface**. With this strategy, query entering depends on the recognizer output to construct the queries. Thus, we propose a derived **panel-assisted interface** which captures user wildcard queries using touch-activated (or tap-activated) interface components (buttons, drop-downs, external keypads). We believe that this interface can reduce input mistakes significantly specially recognition mistakes. In this section we describe the main aspects of the panel-assisted interface, and show screen captures to support the explanation.

5.1 Wildcard Definition

With the panel-assisted interface, the user starts by writing the main query on the canvas. Then, the system extracts all the different alphabetical characters. Any of these characters can become a wildcard. Herein we refer to these characters as *potential wildcards*

The user pops up the input panel with a button with the label *Panel* located at the bottom the left toolbar. The button appears only when the expression in the canvas has at least one potential wildcard. Whenever the panel is opened, interaction with the canvas is disabled, but the user can close the panel at any time.



(a) Drop-down list for potential wildcards

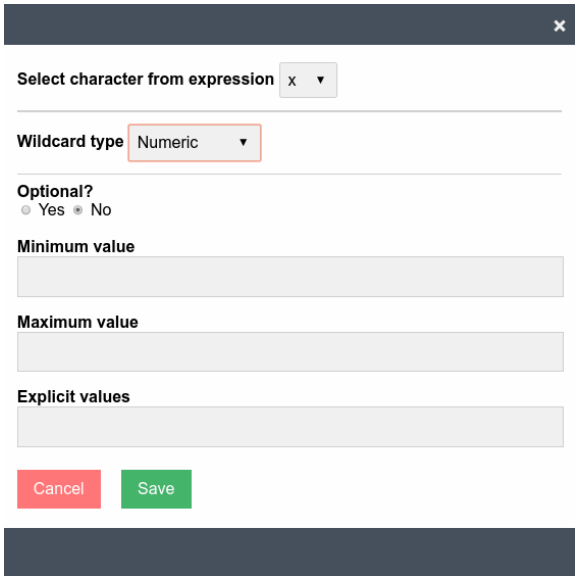
(b) Drop-down list for wildcard type

Fig. 2. Panel-assisted interface for input language

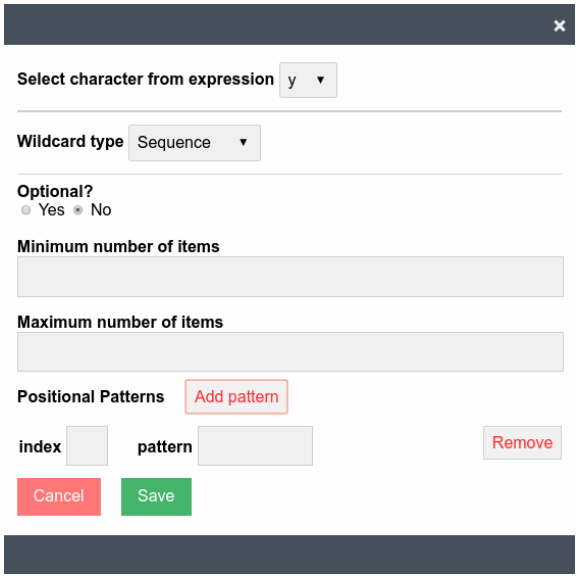
Figure 2 depicts how the wildcard panel updates after the user has written the expression xyz on the canvas. There are two drop-down lists at the top of the panel. One allows the user to switch between the potential wildcards, and the other lets her select the wildcard type. Initially all the potential wildcards are set to type *None*. When the user changes or selects a wildcard type, the panel loads the corresponding attributes that can be parametrized (i.e. the wildcard constraints).

5.2 Wildcard Constraints

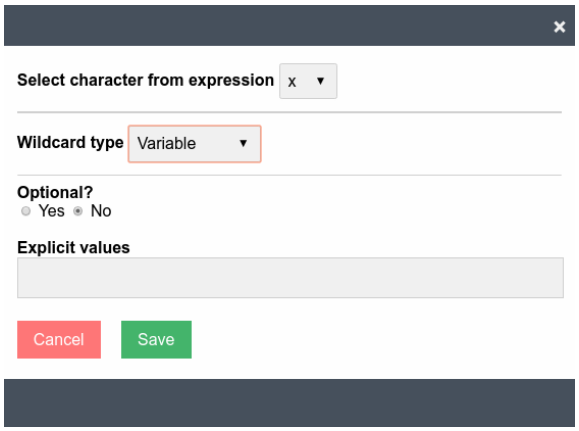
Figure 3 shows how the panel changes based on the wildcard type selected. For each wildcard type, the panel loads a form with the fields that correspond to the wildcard constraints, which can be added or edited by the user. For instance, for the numeric wildcard type the user can enter the `min_val`, `max_val`, and `exp_vals` constraints. The panel also allows specifying when a wildcard is optional through a set of radio buttons (i.e. mutually exclusive selection).



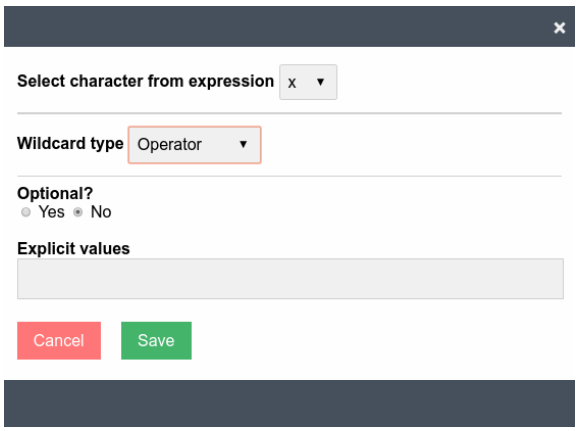
(a) Numeric wildcard



(b) Sequence wildcard

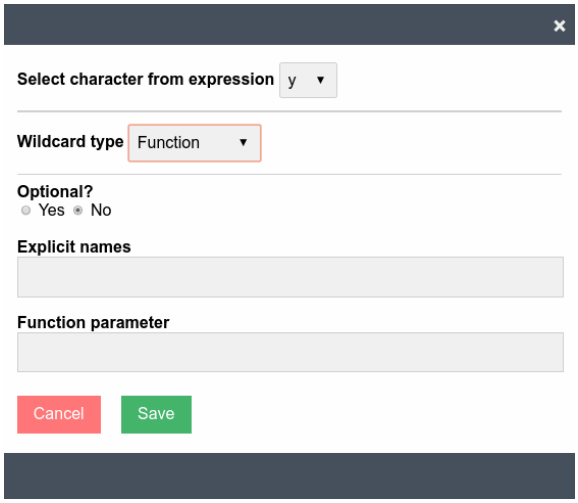


(c) Variable wildcard

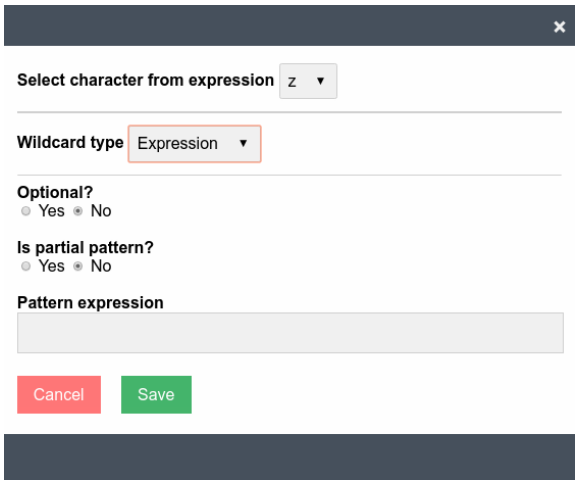


(d) Operator wildcard

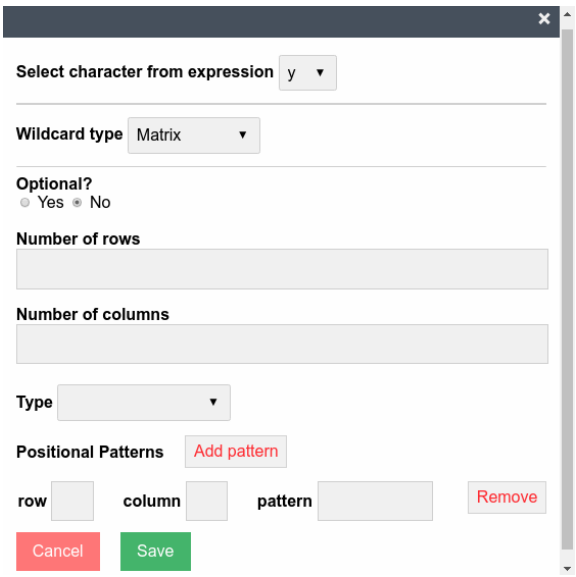
Fig. 3. Wildcard types and their corresponding input fields



(e) Function wildcard



(f) Expression wildcard



(g) Matrix wildcard

Fig. 3. Wildcard types and their corresponding input fields

The majority of the fields in the panel trigger a keypad window when clicked, which allows the user to specify the constraint. Similar to the panel window, whenever the keypad window is displayed the user cannot interact with the rest of the visual interface, although she can close the keypad window at any time.

The keypad window has two parts: (1) an output box that displays the current value, and (2) interactive buttons that allow the user to construct and edit the value. The panel has multiple types of keypads catered towards the different values that constraints can take. All but the naming keypad are based on the traditional calculator

numeric keypad [9, 22, 45]. The naming keypad is based on the QWERTY layout [81]. The keypads also provide visual feedback [64] in two cases: erroneous values (the window displays a message), and the tapping action (the key color changes for a few seconds).

5.3 Keypad Types

- *Decimal keypad.* This keypad allows decimal (positive and negative) values. The constraints that require this keypad are: `min_val` and `max_val` (numeric wildcard).
- *Decimal keypad with support for multiple values.* As the names indicates it, this keypad is an adaptation of the decimal keypad that lets the user specify a list/sequence of comma-separated values. The only constraint that requires this keypad is: `exp_vals` (numeric wildcard).
- *Integer keypad.* This keypad allows positive integers only. The constraints that require this keypad are: `min_items` and `max_items` (numeric wildcard); and `num_rows` and `num_cols` (matrix wildcard).
- *Operator keypad.* This keypad was designed for the constraint `exp_vals` (operator wildcard). In the base interface, the user has to use character identifiers that represent math operators when specifying this constraint (e.g. A represents the addition operator). We use this idea when constructing the string output of this keypad. The options in the keypad correspond to mathematical operators, but the output is constructed using the character identifiers. This keypad allows the user to enter a list/sequence of comma-separated values. For example, if the user enters the input +, x, ÷ the output corresponds to A, M, D.
- *Positional index keypad.* This keypad was designed for the constraint `pos_pat` (sequence and matrix wildcards). The sequence wildcard requires one index for the constraint, whereas the matrix requires two indexes (one for the row and one for the column). This keypad resembles the integer keypad, but it has two additional options to handle the special indexing cases *all* and *one*.
- *Naming keypad.* This keypad was designed for the constraints `exp_vals` (variable wildcard) and `exp_names` (function wildcard). The keypad has the largest set of interactive buttons, composed of numeric digits, the English alphabet, and the Greek alphabet. Mimicking conventional computer keyboards, this keypad has a key `caps` that allows to switch between uppercase and lowercase letters.

The keypads that allow entering multiple values have a key *sep* that stands for *separator* and adds a separator symbol to the output, specifically a comma. All the keypads have the editing keys *del*, *clear*, *cancel*, and *done*. *del* removes the last character in the input. *clear* removes all the characters from the input (i.e. empty value). *done* submits the value. *cancel* closes the keypad window without submitting the value. When the user hits *done*, the system validates if the input is syntactically valid based on the keypad type. If the input is invalid, the keypad cleans the input and displays an error message in the output box for a few seconds, otherwise the value is submitted and the keypad window is closed.

The constraint `pos_pat` (sequence and matrix wildcards) is a peculiar case since a wildcard can have multiple. In the panel, the list of *Positional Patterns* starts empty by default, and the user can add a new pattern with the button *Add pattern*. Any pattern can be removed with the button *Remove* located to the left of the pattern. When adding a new pattern, the system validates if all the previous patterns have been filled, if not it displays a warning message next to the label *Positional Patterns* for a few seconds.

There are some constraints that require the user to enter a handwritten query. These constraints are `param` (function wildcard), `pattern` (expression wildcard), `pos_pat` (sequence and matrix wildcards). For these constraints the panel displays input fields that prompt another instance of the MathBrush canvas when tapped. This second instance has disabled the ability to interact with the wildcard panel. The user needs to write the nested query in this extra canvas and then close the browser window. When the window of the second instance is closed an event captures the query on the extra canvas and sends it to the main instance of MathBrush. Then the wildcard panel displays the nested query in the corresponding input field.

(a) Decimal keypad

(b) Decimal keypad with support for multiple values

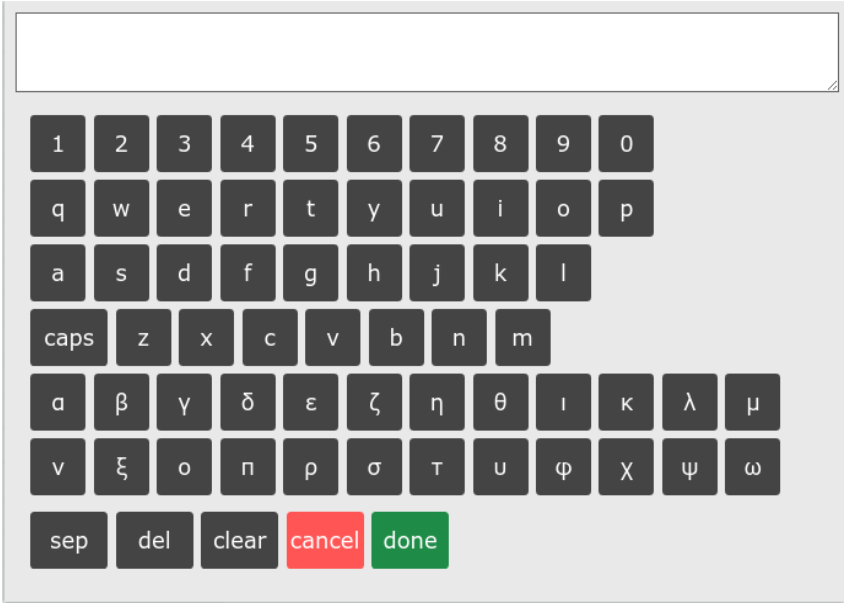
(c) Integer keypad

(d) Operator keypad

(e) Positional index keypad

(f) Keypad with *invalid entry* message

Fig. 4. Different types of keypads in panel-assisted interface



(g) Naming keypad

Fig. 4. Types of keypads in panel-assisted interface

5.4 Wildcard Panel Memory Mechanism

Whenever the user selects a new option in the potential wildcards drop-down changing from `current_option` to `new_option` the flow below is triggered:

- If `new_option` equals `current_option` we stop the flow and no configuration is saved or loaded.
- Otherwise, we save the current configuration for `current_option`. We store the wildcard type, and the constraints values of the potential wildcard. When the potential wildcard type is *None* we simply store the type.
- We load the last stored configuration for `new_option`, setting the value of the input field constraints.

For the wildcard type drop-down, when the user switches to a different type, the panel loads empty input fields constraints corresponding to the new type. The system deletes any data stored for the constraints of the previous type. The system cleans all the data for the wildcard panel whenever the user modifies the query in the canvas.

5.5 Visual Feedback

With the panel-assisted interface, visual feedback still relies on the *explicit language* to provide user feedback of the wildcard query. The feedback is shown whenever the user press the button *Submit* located at the bottom of the panel.

6 QUERY INPUT MISTAKES

We identified and grouped some of the common mistakes that can arise when using the *input language* to enter queries (due to the user handwriting). This is not an exhaustive list.

6.1 Error Feedback

- Missing wildcard definition (i.e. wildcard type not specified): The feedback would display the message `missing type`.
- A character is assigned to multiple wildcard types: The feedback would display the message `multiple types`.
- Wildcard constraint with invalid value: The feedback would display the message `invalid value` for the constraint.
- For a numeric wildcard, `min_val` is greater than `max_val`: The feedback would display the message `invalid bounds` for the wildcard.
- For a sequence wildcard, `min_items` is greater than `max_items`: Same as above.
- The `index` for a constraint `pos_pat` (sequence and expression wildcards) is out of bounds: The feedback would display the message `index out of bounds` for the constraint.
- Circular references in nested queries (function, matrix, and sequence wildcards): For example, in a query where `a` is an expression that uses `b`, and `b` is an expression that uses `a`

```
a = 0
Ea  (b)
Eb  (a)
```

  There is a circular reference between a and b. In this case, the feedback would display the message `circular reference` for wildcard `b`.
- When the system cannot parse a wildcard definition or a wildcard constraint: The feedback would show the message `invalid lines` and then the bracket-enclosed list of line numbers (from the input) that could not be parsed. This case would include mistakes such as wildcard identifier not present in main query, and invalid wildcard type.

6.2 Case Sensitivity

Many of the constraints in the language require the use of uppercase letters, however the system allows the use of lowercase letters. This applies to the following constraints: wildcard type, character IDs for the constraint `exp_vals` in an operator wildcard, placeholder symbol `E` (used to specify an optional wildcard), placeholder symbol `P` for the constraint `pos_pat`, placeholder symbol `P` for the constraint `param` in a function wildcard, and placeholder symbol `P` (used to specify partial matching in a expression wildcard).

6.3 Symbol Similarity

Grissinger [24] posits that some alphanumeric symbols are similar in appearance. Based on commonly confused alphanumeric symbols we propose a few cases to take into account:

- Wildcard types: `0` and uppercase letter `D` are treated as `O` (operator wildcard), `5` and `8` are treated as `S` (sequence wildcard).
- Letter transformations: lowercase letter `l` is treated as `1`; lowercase letter `z` is treated as `2`; uppercase letter `Z` is treated as `7`; uppercase letter `B` is treated as `8`; letters `S` and `s` are treated as `5`; letters `D`, `o`, and `O` are treated as `0`.
- Number transformations: `1` is treated as lowercase letter `l`; `2` is treated as lowercase letter `z`; `7` is treated as uppercase letter `Z`; `8` is treated as uppercase letter `B`; `5` is treated as uppercase letter `S`; `0` is treated as uppercase letter `O`.

- Replacing infinity symbol ∞ with placeholder symbol α: This only applies for the constraints `min_val`, `max_val`, and `pos_pat`.
- Replacing square brackets [ ] with parentheses ( ) for constraints that require an enclosed value (or list of values).
- Lowercase letter `g` and lowercase letter `q`: Any instance of `q` for the constraint `exp_vals` in an operator wildcard is treated as `g`. Similarly, When we have a lowercase letter `c`, we replace it for a lowercase letter `e`.
- Uppercase letter `T` and uppercase letter `I`: Any instance of `I` in a matrix wildcard is treated as `T` (`type` constraint).
- For the `type` constraint in a matrix wildcard, the uppercase letter `O` is replaced with `D` (diagonal matrix).

6.4 Alternatives for Wrong Positioning

- Optional wildcards use the placeholder symbol `E` as a superscript of the wildcard type. The system treats subscript and line cases as valid.
- Function wildcards use the line placeholder symbol `P` for the constraint `param`. The system treats subscript and superscript cases as valid.
- Expression wildcards use the line placeholder symbol `P` to specify partial matching. The system treats subscript and superscript cases as valid.
- Matrix wildcards use the line placeholder symbol `T` for the constraint `type`. The system treats subscript and superscript cases as valid.

To illustrate how the feedback would behave in the presence of some of the above cases we offer an example. First let us consider the Fourier Sine and Cosine Series:

$$f(x) = a_0 + \sum_{n=1}^{\infty}(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L})$$

Below we have a query that will match this expression but also has some input mistakes

$$f(x) = a_0 s \sum_{n=1}^{\infty} c$$

$$0s \quad (A - s - pM - q_C)$$

$$EcP \quad (a_n z + b_n e)$$

$$fz^P \quad (m)$$

$$fe_p \quad (q)$$

$$EmP \quad (n\pi x)$$

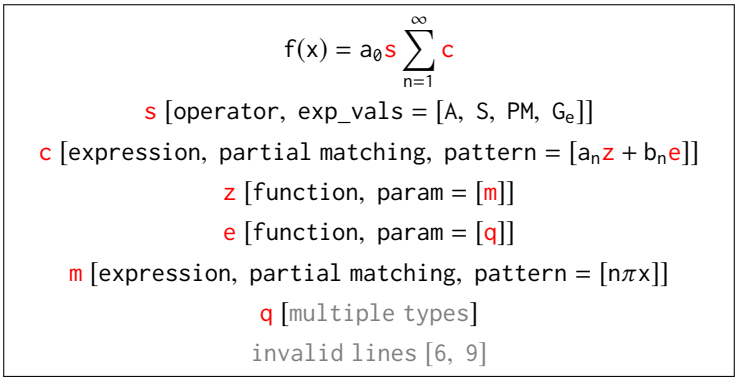
$$hT^{SD}$$

$$Eq$$

$$Mq$$

$$Hq \quad (1 - 2)$$

The visual feedback will render the following



7 EXPERIMENTAL FRAMEWORK

The input language imposes a particular structure on the user input. We argue that the difference between our two interfaces is related to the trade-off between query recognition accuracy and ease of input. The base interface captures the query in a more natural way, needing only user handwriting whereas the panel-assisted interface can reduce significantly recognition mistakes. Both interfaces still impose some degree of formality when writing queries, which is tied to the use of the input language. Conducting a usability study is outside of the scope of this project. However, we present an experimental framework that can be used as a base to carry out such experiments.

7.1 Research Questions

First, we want to explore users' perception regarding the formality of the input language. The concept of **structuration theory** [23, 32, 60] states that *structure enables action by providing a guide, but can also constrain when the guide overly constricts expressibility*. Using this idea, we propose multiple questions.

- How the level of formality in the input language affects the expressibility of the language?
- Do users consider that the input language is formal enough that allows them to construct their desired wildcard queries?
- Do users consider that the input language overly constricts user expressibility?
- How much effort do users require to learn and use the input language in order to construct wildcard queries?

Second, we want to explore users perception regarding the usefulness of the explicit language which is used to provide visual feedback. A consideration here is that the structure of the language itself and its visualization/presentation are strongly tied (at least from the user point of view). We propose the following questions:

- Can the users understand the textual information presented in the visual feedback?
- Do users consider the visual feedback helpful?
- Do users understand the meaning behind the different coloring techniques used in the feedback?
- Do users understand how errors and warnings are presented in the feedback?

Lastly, we want to compare the base interface and the panel-assisted interface and whether there is a clear preference for users. To assess the performance of an experiment we propose to focus on four variables: correctness, duration (time needed to complete the task), use of help, use of retries. Some of these variables have been used in other experimental studies [18, 32, 73]. We expect for the panel-assisted interface to achieve better correctness, to require less use of help, and to require less retries.

7.2 Experimental Methodology

For the methodology we want to focus on the last route mentioned above, the comparison between the two interfaces. We propose a within-subject experiment [7] which allows collecting a greater set of data points and reduces bias and the distortion of results [18, 32, 73].

Based on the procedure presented by Kaufmann and Bernstein [32], we propose the following overview of the experimental procedure for each participant.

7.3 Experimental Procedure

(1) Read introductory notes of experiment
(2) Read instructions on the input language and the explicit language
(3) See examples of wildcard queries constructed with the panel-assisted interface
(4) Participate in free session to interact with the panel-assisted interface for 5 minutes
(5) Enter each query from the experimental set with the panel-assisted interface. One at the time, cleaning the canvas after each query
(6) Fill out usability questionnaire for the panel-assisted interface
(7) Proceed by repeating steps (3)-(6) with the base interface
(8) Fill out comparison questionnaire
(9) Fill out demographic questionnaire

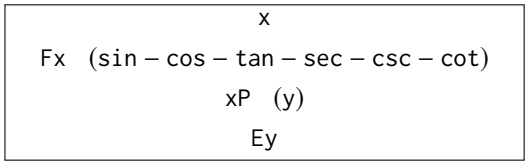
Participants use the panel-assisted interface first in order to account for learning bias.

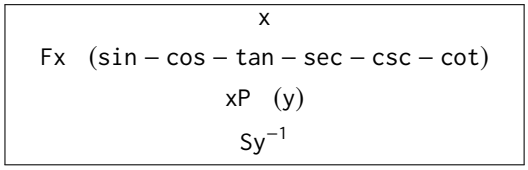
For the usability questionnaire, the study could use the *System Usability Scale* (SUS) as a base. Proposed by Brooke [11], SUS is is a standardized usability test that constains 10 questions, where each question uses a Likert scale to measure a participant's impression regarding the interface.

For the comparison questionnaire, the goal would be to determine if the participant prefers one interface over the other and in what degree, is it for all the aspects of the interface?

The experimental set refers to the task set used in the experiment. Each task would correspond to a textual description, and the goal is to construct a wildcard query that fulfills the description. Below we present multiple examples of tasks and queries that fulfill the tasks.

- *Description: Please write a query to find all the trigonometric functions with just one parameter.*
  For this task two different valid queries are:

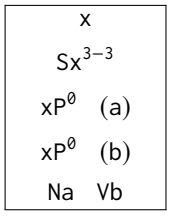


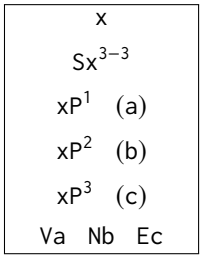


- *Description: Please write a query to find sequences with exactly 3 items. One of the items is a number. One of the items is a variable.*

[7] A within-subject design is a type of experimental design in which all participants are exposed to every treatment or condition.

For this task two different valid queries are:





The user does not really need to specify the constraints for the third item in the sequence.

7.4 Evaluation Metrics

To evaluate the experiment and analyze the data collected we propose multiple variables of interest:

- **Correctness**. Correctness measures the degree of success of the participant when completing the tasks. After completing each task, a facilitator should assess the correctness and record this value. If treated as a quantitative metric, it would be possible to compute a cumulative value for each participant (e.g. mean).
- **Duration**. The duration refers to how much time the participant spent performing the tasks. To assure an upper limit in terms of study duration, we could establish a time limit for each task, that if reached forces the participant to advance to the next task. For example, the limit could be 5 minutes.
- **Use of help**. During the experiment the participant would receive instructions on how the explicit language and the input language work. These instructions could be in a handout or it could be a browser tab in the same device used to complete the tasks. We could record metrics such as: number of times using help information, duration of each time, whether accessing the help information affected participant progress in the task.
- **Use of retries**. In MathBrush the user can clean the entire canvas at any time and start from scratch. It might be interesting to capture how the user behaves with this feature. For example, a higher number of retries could indicate some frustration from the participant because she does not comprehend how to construct the queries.
- **Usability** and **Learnability**. These two variables would be captured with the post-questionnaires filled out by the participant after using each interface. We could contrast these responses with the above metrics to determine if there is some correlation. For example, a participant that obtained better correctness scores, could be more likely to give more positive scores in the questionnaire. Usability refers to the participant opinion on how well each interface helped her to complete the tasks, whereas learnability would focus on the participant's interaction with the grammar required by the input language.

8 DISCUSSION

In this paper we explore the use of wildcards for math retrieval in the context of handwritten math and pen-based interfaces. We found that the majority of existing pen-based handwritten math systems offer limited support for the use of wildcards when formulating queries. We also examine the concept of math formula query languages, which take advantage of the use of wildcards to allow for more expressiveness in the creation of queries. We compare three existing query languages using multiple quality attributes. None of the languages is completely suitable to integrate with handwritten math interfaces, thus we decided to focus on this problem. For that, we propose a novel query language aimed to help users construct wildcard queries. As a proof of concept, we implemented our language on MathBrush, a pen-based mathematical system currently maintained by the Symbolic Computation Group (SGC) at the University of Waterloo. One of the main challenges that handwritten math systems face is the proper recognition of user input. To name a few of the causes: different writing styles, similarity between symbols, touching symbols, messy handwriting, spacing issues. Motivated by this, we developed two interfaces that explore the trade-off between query accuracy and ease of input. Due to the scope of this project, we did not run any experimental evaluation. However, we propose an experimental framework with multiple courses of action that can be used to evaluate the interfaces.

Besides the experimental evaluation, another future direction is to apply the concept of wildcard language to the information retrieval process. The goal would be to allow matching/similarity relaxation. The user would input a query and a set of guidelines on how to rank search results, which would be used to enhance the final ranking. For example, let us say that for each guideline the user assigns a rating between 0 and 10; we have the query $\sqrt{a+b} = 10$ and the relaxations (1) exact match has a rating of 10 (highest priority), (2) replacing constants by numeric wildcards has a rating of 8, (3) replacing character symbols by variable wildcards has a rating of 6, and (4) replacing character symbols by expression wildcards has a rating of 5. The system could create a tree of potential relaxations, each one associated with a rating, which measures the preference of the relaxation over the rest.

ACKNOWLEDGMENTS

Firstly, thank you very much to my supervisor: Frank Tompa. Attending Waterloo has been a wonderful experience, this journey took 2 years+ but sometimes seems like yesterday when I started. Frank, I really appreciate your support over this time. The flexibility you provided allowed me to explore different research areas and also to experiment firsthand industry software engineering thanks to the Co-op program.

I also want to thank George Labahn and Mirette Marzouk. George, our initial discussions exploring the viability of using MathBrush for the prototype, as well as your feedback during the revision process are highly appreciated. Mirette, all of your help and patience during the implementation process was really valuable.

A very special thanks to my friends Elena Quan, Yerbol Aussat, Henry Chen, Samin Riasat, and Shahid Khaliq. I was incredibly lucky to have shared great memories with all of you.

Lastly, I want to acknowledge the financial support of my studies provided by NSERC (Natural Sciences and Engineering Council of Canada) and the University of Waterloo.

REFERENCES

[1] Milton Abramowitz and Irene A Stegun. 1965. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*. Vol. 55. Courier Corporation.

[2] Karteek Alahari, Satya Lahari Putrevu, and CV Jawahar. 2006. Learning mixtures of offline and online features for handwritten stroke recognition. In *18th International Conference on Pattern Recognition (ICPR'06)*, Vol. 3. IEEE, 379–382.

[3] Moody Ebrahem Altamimi and Abdou Youssef. 2008. A math query language with an expanded set of wildcards. *Mathematics in Computer Science* 2, 2 (2008), 305–331.

[4] Francisco Alvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. 2016. An integrated grammar-based approach for mathematical expression recognition. *Pattern Recognition* 51 (2016), 135–147.

[5] Robert H Anderson. 1967. Syntax-directed recognition of hand-printed two-dimensional mathematics. In *Symposium on Interactive Systems for Experimental Applied Mathematics: Proceedings of the Association for Computing Machinery Inc. Symposium*. ACM, 436–459.

[6] Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. 2004. A content based mathematical search engine: Whelp. In *International Workshop on Types for Proofs and Programs*. Springer, 17–32.

[7] Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. 2014. A global learning approach for an online handwritten mathematical expression recognition system. *Pattern Recognition Letters* 35 (2014), 68–77.

[8] Olivier Bau and Wendy E Mackay. 2008. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. ACM, 37–46.

[9] Francesco Bertelli. 2011. A brief history of the numeric keypad. Retrieved Jul 17, 2019 from https://uxdesign.cc/a-brief-history-of-the-numeric-keypad-59112cbf4c49

[10] Andrew Bragdon, Robert Zeleznik, Brian Williamson, Timothy Miller, and Joseph J LaViola Jr. 2009. GestureBar: improving the approachability of gesture-based interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2269–2278.

[11] John Brooke et al. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.

[12] Mehmet Celik and Berrin Yanikoglu. 2011. Probabilistic mathematical formula recognition using a 2d context-free graph grammar. In *2011 International Conference on Document Analysis and Recognition*. IEEE, 161–166.

[13] Mark Ciampa. 2013. A comparison of password feedback mechanisms and their impact on password entropy. *Inf. Manag. Comput. Security* 21, 5 (2013), 344–359. https://doi.org/10.1108/IMCS-12-2012-0072

[14] Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. 2009. From coding theory to efficient pattern matching. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 778–784.

[15] Supervising Committee, Dr. Bo Yuan Reader, David Stalnaker B. S, David Stalnaker M. S, and Advisor Dr. Richard Zanibbi. 2013. Math Expression Retrieval Using Symbol Pairs in Layout Trees.

[16] Pieter De Decker. 2014. *Designing Intelligible Visualizations To Accommodate Mid-Air Gesture Input*. Master's thesis. tUL.

[17] Lanfang Dong and Hanchao Liu. 2017. Recognition of Offline Handwritten Mathematical Symbols Using Convolutional Neural Networks. In *Image and Graphics - 9th International Conference, ICIG 2017, Shanghai, China, September 13-15, 2017, Revised Selected Papers, Part I*. 149–161. https://doi.org/10.1007/978-3-319-71607-7_14

[18] Ashley G Durham and Henry H Emurian. 1998. Learning and retention with a menu and a command line interface. *Computers in human behavior* 14, 4 (1998), 597–620.

[19] Eventra. [n. d.]. Welcome to MoboMath. Retrieved Feb 16, 2019 from http://www.enventra.com/doc/mobomath/index.html

[20] Leah Findlater and Joanna McGrenere. 2004. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 89–96.

[21] Connor Flood. 2017. *MathBrush web application: Design and implementation of an online pen-input interface for computer algebra systems*. Master's thesis. University of Waterloo.

[22] Rick Furr. 2013. Keyboard Trivia. Retrieved Jul 17, 2019 from http://www.vcalc.net/Keyboard.htm

[23] Anthony Giddens. 1984. *The constitution of society: Outline of the theory of structuration*. Univ of California Press.

[24] Matthew Grissinger. 2012. Avoiding confusion with alphanumeric characters. *P & T : a peer-reviewed journal for formulary management* 37, 12 (Dec 2012), 663–665. https://www.ncbi.nlm.nih.gov/pubmed/23319841 23319841[pmid].

[25] Tovi Grossman, Ken Hinckley, Patrick Baudisch, Maneesh Agrawala, and Ravin Balakrishnan. 2006. Hover widgets: using the tracking state to extend the capabilities of pen-operated devices. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 861–870.

[26] Wei Guo, Wei Su, Lian Li, Ning An, and Linwei Cui. 2011. MQL: a mathematical formula query language for mathematical search. In *2011 14th IEEE International Conference on Computational Science and Engineering*. IEEE, 245–250.

[27] Lei Hu and Richard Zanibbi. 2016. MST-based Visual Parsing of Online Handwritten Mathematical Expressions. In *15th International Conference on Frontiers in Handwriting Recognition, ICFHR 2016, Shenzhen, China, October 23-26, 2016*. 337–342. https://doi.org/10.1109/ICFHR.2016.0070

[28] Xuan Hu, Liangcai Gao, Xiaoyan Lin, Zhi Tang, Xiaofan Lin, and Josef B Baker. 2013. Wikimirs: a mathematical information retrieval system for wikipedia. In *Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 11–20.

[29] Frank Julca-Aguilar. 2016. *Recognition of online handwritten mathematical expressions using contextual information*. Ph.D. Dissertation. Université de Nantes; Université Bretagne Loire; Universidade de São Paulo.

[30] Frank D. Julca-Aguilar. 2016. *Recognition of Online Handwritten Mathematical Expressions using Contextual Information. (Reconnaissance d'Expressions Mathématiques Manuscrites Guidéepar le Contexte)*. Ph.D. Dissertation. Université Bretagne Loire, France. https://tel.archives-ouvertes.fr/tel-01326354

[31] Shahab Kamali and Frank Wm. Tompa. 2010. A new mathematics retrieval system. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, Jimmy Huang, Nick Koudas, Gareth J. F. Jones, Xindong Wu, Kevyn Collins-Thompson, and Aijun An (Eds.). ACM, 1413–1416. https://doi.org/10.1145/1871437.1871635

[32] Esther Kaufmann and Abraham Bernstein. 2010. Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases. *Web Semantics: Science, Services and Agents on the World Wide Web* 8, 4 (2010), 377–393.

[33] Michael Kohlhase, Stefan Anca, Constantin Jucovschi, Alberto González Palomo, and Ioan A Sucan. 2008. MathWebSearch 0.4, a semantic search engine for mathematics. *Manuscript at http://mathweb. org/projects/mws/pubs/mkm08. pdf* (2008).

[34] George Labahn, Edward Lank, Scott MacLean, Mirette S. Marzouk, and David Tausky. 2008. MathBrush: A System for Doing Math on Pen-Based Devices. In *The Eighth IAPR International Workshop on Document Analysis Systems, DAS 2008, September 16-19, 2008, Nara, Japan*, Koichi Kise and Hiroshi Sako (Eds.). IEEE Computer Society, 599–606. https://doi.org/10.1109/DAS.2008.21

[35] George Labahn, Edward Lank, Mirette S. Marzouk, Andrea Bunt, Scott MacLean, and David Tausky. 2008. MathBrush: A Case Study for Pen-based Interactive Mathematics. In *Sketch Based Interfaces and Modeling, Annecy, France, 2008. Proceedings*, Christine Alvarado and Marie-Paule Cani (Eds.). Eurographics Association, 143–150. https://doi.org/10.2312/SBM/SBM08/143-150

[36] Stefan Langer and Jöran Beel. 2017. Apache Lucene as Content-Based-Filtering Recommender System: 3 Lessons Learned. In *Proceedings of the Fifth Workshop on Bibliometric-enhanced Information Retrieval (BIR) co-located with the 39th European Conference on Information Retrieval (ECIR 2017), Aberdeen, UK, April 9th, 2017*. 85–92. http://ceur-ws.org/Vol-1823/paper8.pdf

[37] Joseph J LaViola. 2011. Mathematical sketching: An approach to making dynamic illustrations. In *Sketch-based Interfaces and Modeling*. Springer, 81–118.

[38] Joseph J LaViola Jr. 2007. An initial evaluation of MathPad2: A tool for creating dynamic mathematical illustrations. *Computers & Graphics* 31, 4 (2007), 540–553.

[39] Joseph J LaViola Jr, Anamary Leal, Timothy S Miller, and Robert C Zeleznik. 2008. Evaluation of techniques for visualizing mathematical expression recognition results. In *Proceedings of graphics interface 2008*. Canadian Information Processing Society, 131–138.

[40] Chuanjun Li, Timothy S Miller, Robert C Zeleznik, and Joseph J LaViola Jr. 2008. AlgoSketch: Algorithm Sketching and Interactive Computation.. In *SBM*. Citeseer, 175–182.

[41] Peiyu Li, Manuel Bouillon, Eric Anquetil, and Grégoire Richard. 2013. User and system cross-learning of gesture commands on pen-based devices. In *IFIP Conference on Human-Computer Interaction*. Springer, 337–355.

[42] Paul Libbrecht and Erica Melis. 2006. Methods to access and retrieve mathematical content in activemath. In *International Congress on Mathematical Software*. Springer, 331–342.

[43] Martin Líška, Petr Sojka, Michal Růžička, and Petr Mravec. [n. d.]. Web Interface and Collection for Mathematical Retrieval: WebMIaS and MREC. In *Towards a Digital Mathematics Library.*, Petr Sojka and Thierry Bouche (Eds.).

[44] Daniel W Lozier. 2001. The NIST Digital Library of Mathematical Functions Project. In *IN ANNALS OF MATHEMATICS AND ARTIFCIAL INTELLIGENCE*. Citeseer.

[45] Mary Champion Lutz and Alphonse Chapanis. 1955. Expected locations of digits and letters on ten-button keysets. *Journal of Applied Psychology* 39, 5 (1955), 314.

[46] Scott MacLean and George Labahn. 2013. A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. *International Journal on Document Analysis and Recognition (IJDAR)* 16, 2 (2013), 139–163.

[47] Diane Marinkas, Robert C Zeleznik, and Joseph J LaViola Jr. 2009. Shadow buttons: exposing wimp functionality while preserving the inking surface in sketch-based interfaces. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*. ACM, 159–164.

[48] Nicholas Elias Matsakis. 1999. *Recognition of handwritten mathematical expressions*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[49] Osama A. Mehdi, Hamidah Ibrahim, Lilly Suriani Affendey, Eric Pardede, and Jinli Cao. 2018. Exploring instances for matching heterogeneous database schemas utilizing Google similarity and regular expression. *Comput. Sci. Inf. Syst.* 15, 2 (2018), 295–320. http://doiserbia.nb.rs/Article.aspx?id=1820-02141800002M

[50] Alexandra Mendes. 2012. *Structured editing of handwritten mathematics*. Ph.D. Dissertation. University of Nottingham, UK. http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.576155

[51] Alexandra Mendes, Roland Carl Backhouse, and João Fernando Ferreira. 2014. Structure Editing of Handwritten Mathematics: Improving the Computer Support for the Calculational Method. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops*

and Surfaces, ITS 2014, Dresden, Germany, November 16 - 19, 2014, Raimund Dachselt, T. C. Nicholas Graham, Kasper Hornbæk, and Miguel A. Nacenta (Eds.). ACM, 139–148. https://doi.org/10.1145/2669485.2669495

[52] Bruce R Miller and Abdou Youssef. 2003. Technical aspects of the digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence* 38, 1-3 (2003), 121–136.

[53] Fan Min, Xindong Wu, and Zhenyu Lu. 2009. Pattern matching with independent wildcard gaps. In *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 194–199.

[54] Jozef Mišutka and Leo Galamboš. 2011. System description: Egomath2 as a tool for mathematical searching on wikipedia. org. In *International Conference on Intelligent Computer Mathematics*. Springer, 307–309.

[55] Harold Mouchère, Richard Zanibbi, Utpal Garain, and Christian Viard-Gaudin. 2016. Advancing the state of the art for handwritten math recognition: the CROHME competitions, 2011-2014. *IJDAR* 19, 2 (2016), 173–189. https://doi.org/10.1007/s10032-016-0263-5

[56] Rajesh Munavalli and Robert Miner. 2006. Mathfind: a math-aware search engine. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 735–735.

[57] Matei Negulescu, Jaime Ruiz, and Edward Lank. 2010. Exploring usability and learnability of mode inferencing in pen/tablet interfaces. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*. Eurographics Association, 87–94.

[58] Hai Dai Nguyen, Anh Duc Le, and Masaki Nakagawa. 2016. Recognition of Online Handwritten Math Symbols Using Deep Neural Networks. *IEICE Transactions* 99-D, 12 (2016), 3110–3118. https://doi.org/10.1587/transinf.2016EDP7102

[59] Tam T Nguyen, Kuiyu Chang, and Siu Cheung Hui. 2012. A math-aware search engine for math question answering system. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 724–733.

[60] Wanda J Orlikowski. 1992. The duality of technology: Rethinking the concept of technology in organizations. *Organization science* 3, 3 (1992), 398–427.

[61] Yongki Park, Jaehoon Kim, and Kyogu Lee. 2015. Effects of Auditory Feedback on Menu Selection in Hand-Gesture Interfaces. *IEEE MultiMedia* 22, 1 (2015), 32–40. https://doi.org/10.1109/MMUL.2015.5

[62] Nidhin Pattaniyil and Richard Zanibbi. 2014. Combining TF-IDF Text Retrieval with an Inverted Index over Symbol Pairs in Math Expressions: The Tangent Math Search Engine at NTCIR 2014. In *Proceedings of the 11th NTCIR Conference on Evaluation of Information Access Technologies, NTCIR-11, National Center of Sciences, Tokyo, Japan, December 9-12, 2014*. http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings11/pdf/NTCIR/Math-2/08-NTCIR11-MATH-PattaniyilN.pdf

[63] Roberto M. Pinheiro Pereira, Caio Eduardo Falcao Matos, Geraldo Braz Junior, João Dallyson Sousa de Almeida, and Anselmo Cardoso de Paiva. 2016. A Deep Approach for Handwritten Musical Symbols Recognition. In *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web, Webmedia 2016, Teresina, Piauí State, Brazil, November 8-11, 2016*. 191–194. https://doi.org/10.1145/2976796.2988171

[64] Jonathan Perrinet, Xabiel G Pañeda, Sergio Cabrero, David Melendi, Roberto García, and Víctor García. 2011. Evaluation of virtual keyboards for interactive digital television applications. *International Journal of Human-Computer Interaction* 27, 8 (2011), 703–728.

[65] Raihan Ur Rasool, Maleeha Najam, Hafiz Farooq Ahmad, Hua Wang, and Zahid Anwar. 2019. A novel JSON based regular expression language for pattern matching in the internet of things. *J. Ambient Intelligence and Humanized Computing* 10, 4 (2019), 1463–1481. https://doi.org/10.1007/s12652-018-0869-1

[66] Steve Smithies, Kevin Novins, and James Arvo. 1999. A handwriting-based equation editor. In *Graphics Interface*, Vol. 99. 84–91.

[67] Steve Smithies, Kevin Novins, and James Arvo. 2001. Equation entry and editing via handwriting and gesture recognition. *Behaviour & information technology* 20, 1 (2001), 53–67.

[68] Petr Sojka and Martin Líška. 2011. The Art of Mathematics Retrieval. In *Proceedings of the ACM Conference on Document Engineering, DocEng 2011*. Association of Computing Machinery, Mountain View, CA, 57–60. https://doi.org/10.1145/2034691.2034703

[69] David Stalnaker and Richard Zanibbi. 2015. Math expression retrieval using an inverted index over symbol pairs. In *Document Recognition and Retrieval XXII, San Francisco, California, USA, February 11-12, 2015*. 940207. https://doi.org/10.1117/12.2074084

[70] G Sudeepthi, G Anuradha, and M Surendra Prasad Babu. 2012. A survey on semantic web search engine. *International Journal of Computer Science Issues (IJCSI)* 9, 2 (2012), 241.

[71] symbolab 2017. Symbolab About. Retrieved May 14, 2019 from https://www.symbolab.com/about

[72] Ernesto Tapia and Raúl Rojas. 2003. Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In *International workshop on graphics recognition*. Springer, 329–340.

[73] Susan Wiedenbeck. 1999. The use of icons and labels in an end user application program: an empirical study of learning and retention. *Behaviour & Information Technology* 18, 2 (1999), 68–82.

[74] Richard Zanibbi and Dorothea Blostein. 2012. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)* 15, 4 (2012), 331–357.

[75] Richard Zanibbi, Dorothea Blostein, and James R Cordy. 2001. Baseline structure analysis of handwritten mathematics notation. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*. IEEE, 768–773.

[76] Richard Zanibbi, Dorothea Blostein, and James R. Cordy. 2002. Recognizing mathematical expressions using tree transformation. *IEEE Transactions on pattern analysis and machine intelligence* 24, 11 (2002), 1455–1467.

[77] Richard Zanibbi, Kenny Davila, Andrew Kane, and Frank Wm. Tompa. 2015. The Tangent Search Engine: Improved Similarity Metrics and Scalability for Math Formula Search. *CoRR* abs/1507.06235 (2015). arXiv:1507.06235 http://arxiv.org/abs/1507.06235

[78] Richard Zanibbi, Kenny Davila, Andrew Kane, and Frank Wm. Tompa. 2016. Multi-Stage Math Formula Search: Using Appearance-Based Similarity Metrics at Scale. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*. 145–154. https://doi.org/10.1145/2911451.2911512

[79] Richard Zanibbi and Awelemdy Orakwue. 2015. Math Search for the Masses: Multimodal Search Interfaces and Appearance-Based Retrieval. In *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*. 18–36. https://doi.org/10.1007/978-3-319-20615-8_2

[80] Robert Zeleznik, Timothy Miller, Chuanjun Li, and Joseph J LaViola. 2008. Mathpaper: Mathematical sketching with fluid support for interactive computation. In *International Symposium on Smart Graphics*. Springer, 20–32.

[81] Shumin Zhai, Michael Hunter, and Barton A Smith. 2000. The Metropolis Keyboard??? An Exploration of Quantitative Techniques for Virtual Keyboard Design. In *in UIST'2000: Proceedings of the 13th annual ACM symposium on User interface software and technology. 2000*. Citeseer.

[82] Shumin Zhai, Per Ola Kristensson, Caroline Appert, Tue Haste Anderson, Xiang Cao, et al. 2012. Foundational issues in touch-surface stroke gesture design — an integrative review. *Foundations and Trends® in Human–Computer Interaction* 5, 2 (2012), 97–205.

[83] Jianshu Zhang, Jun Du, Shiliang Zhang, Dan Liu, Yulong Hu, Jin-Shui Hu, Si Wei, and Li-Rong Dai. 2017. Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition. *Pattern Recognition* 71 (2017), 196–206. https://doi.org/10.1016/j.patcog.2017.06.017

[84] Ling Zhang, Dorothea Blostein, and Richard Zanibbi. 2005. Using fuzzy logic to analyze superscript and subscript relations in handwritten mathematical expressions. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE, 972–976.

[85] Ting Zhang. 2017. *New Architectures for Handwritten Mathematical Expressions Recognition. (Nouvelles architectures pour la reconnaissance des expressions mathématiques manuscrites)*. Ph.D. Dissertation. University of Nantes, France. https://tel.archives-ouvertes.fr/tel-01754478

[86] Ting Zhang, Harold Mouchère, and Christian Viard-Gaudin. 2016. Online Handwritten Mathematical Expressions Recognition by Merging Multiple 1D Interpretations. In *15th International Conference on Frontiers in Handwriting Recognition, ICFHR 2016, Shenzhen, China, October 23-26, 2016*. 187–192. https://doi.org/10.1109/ICFHR.2016.0045

[87] Xu-Yao Zhang, Yoshua Bengio, and Cheng-Lin Liu. 2017. Online and offline handwritten chinese character recognition: A comprehensive study and new benchmark. *Pattern Recognition* 61 (2017), 348–360.

[88] Wei Zhong. [n. d.]. Approach0. A math-aware search engine. Retrieved May 14, 2019 from https://github.com/approach0/search-engine