

## MIT Open Access Articles

*Constructing Embodied Algebra by Sketching*

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

**Citation:** Saquib, Nazmus, Kazi, Rubaiat, Wei, Li-yi, Mark, Gloria and Roy, Deb. 2021. "Constructing Embodied Algebra by Sketching."

**Persistent URL:** <https://hdl.handle.net/1721.1/146008>

**Version:** Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

**Terms of use:** Creative Commons Attribution 4.0 International license



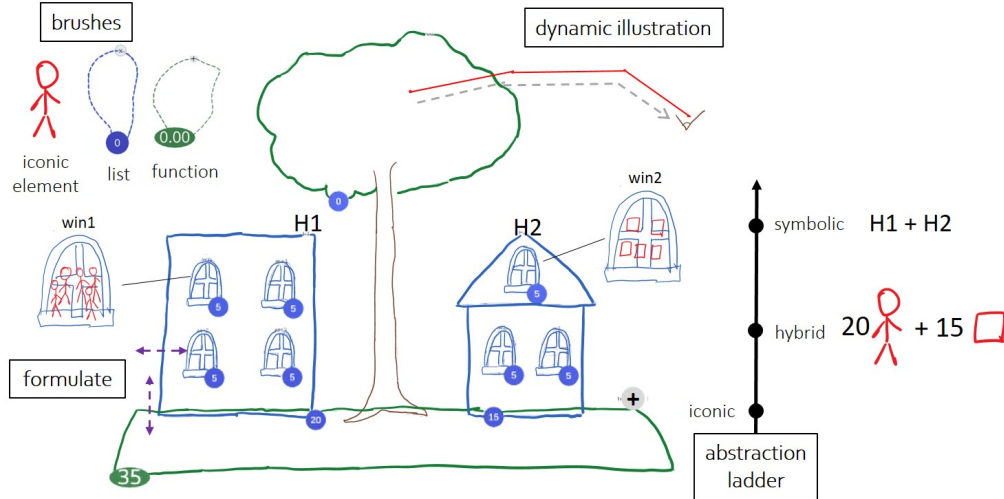
# Constructing Embodied Algebra by Sketching

Nazmus Saquib  
MIT Media Lab,  
United States  
saquib@mit.edu

Rubaiat Habib Kazi  
Li-Yi Wei  
Adobe Research, United  
States  
rubaiat.habib@gmail.com  
review@liyiwei.org

Gloria Mark  
Informatics Department  
University of California,  
Irvine, United States  
gmark@uci.edu

Deb Roy  
MIT Media Lab,  
United States  
dkroy@media.mit.edu



**Figure 1:** Noyon combines sketching and computer algebra algorithms to construct mathematical expressions using iconic (sketch) elements. In this example, the population of two houses are being added. There are three unique *brushes* (iconic elements, list, function) to construct mathematical expressions in the form of hierarchical compositions, grounding algebraic expressions in hand-drawn sketches. The houses named H1 and H2 are lists, and the street containing them is an addition function. Iconic elements (in red) populate the windows (which are also lists). Bi-directional *abstraction ladders* allow exploring each composition in three layers of mathematical understanding (iconic, hybrid expression, symbolic). Path animation capabilities and moving in/out gestures allow children structures to dynamically change membership between parent compositions according to a user's visual preference, which allows us to *formulate* different algebraic expressions. Finally, *dynamic illustration* capabilities enhance Noyon's fluid nature of sketching, storytelling, and presentations.

## ABSTRACT

Mathematical models and expressions traditionally evolved as symbolic representations, with cognitively arbitrary rules of symbol manipulation. The embodied mathematics philosophy posits that abstract math concepts are layers of metaphors grounded in our intuitive arithmetic capabilities, such as categorizing objects and part-whole analysis. We introduce a design framework that facilitates the construction and exploration of embodied representations for algebraic expressions, using interactions inspired by innate arithmetic capabilities. We instantiated our design in a sketch interface that enables construction of visually interpretable compositions that are directly mappable to algebraic expressions and explorable

through a ladder of abstraction [47]. The emphasis is on bottom-up construction, with the user sketching pictures while the system generates corresponding algebra. We present diverse examples created by our prototype system. A coverage of the US Common Core curriculum and playtesting studies with children point to the future direction and potential for a sketch-based design paradigm for mathematics.

## CCS CONCEPTS

• **Human-centered computing** → **Graphical user interfaces; Interaction design theory, concepts and paradigms.**

## KEYWORDS

sketching, math modeling, embodied math, embodied cognition

## ACM Reference Format:

Nazmus Saquib, Rubaiat Habib Kazi, Li-Yi Wei, Gloria Mark, and Deb Roy. 2021. Constructing Embodied Algebra by Sketching. In *CHI Conference on Human Factors in Computing Systems (CHI '21)*, May 8–13, 2021, Yokohama, Japan. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3411764.3445460>



This work is licensed under a Creative Commons Attribution-Share Alike International 4.0 License.

CHI '21, May 8–13, 2021, Yokohama, Japan  
© 2021 Copyright held by the owner/author(s).  
ACM ISBN 978-1-4503-8096-6/21/05.  
<https://doi.org/10.1145/3411764.3445460>

## 1 INTRODUCTION

*“I now view the effort to find a fundamental theory of physics as in many ways another challenge in language design — perhaps even the ultimate such challenge. Designing a computational language is to create a bridge between two domains: the abstract world of what is possible to do, and the “mental” world of what people understand and are interested in doing. The challenge is [...] to give people a way to describe these [abstractions].”* - Stephen Wolfram [52].

Mathematics as a discipline has relied on symbolic abstractions and symbol manipulations for the last few centuries. The power and hierarchical flexibility of symbols to do mathematics at higher dimensions and dealing with abstract concepts is undeniable. Algebra has been widely used to manipulate symbols for abstract mathematics. However, such abstraction is not always intuitive, and creates a barrier to explanation, exploration, and understanding [45].

We often start with pictorial or manipulable objects to construct mathematical concepts in an explainable manner, grounded in intuitive and embodied interactions such as counting, grouping, sets, compositions of sets, etc. (Figure 2). We then move on to learn symbolic abstraction (numerals, named variables), manipulation, and eventually Computer Algebra Systems (CAS) in order to explore with abstractions faster, and train ourselves to translate different scenarios to such abstractions through parameterization (e.g., learn to extract parameters from a visual scene and form algebraic relationships between them).

In practice, many learners lose the natural, intuitive underpinnings of those abstractions and resort to memorizing rules of manipulation (“symbol pushing”) which become cognitively arbitrary and severed from their intuitive grounding [45]. Current CAS software (such as [9, 14, 31–33]) only provide notebook interfaces where code and plots can be combined, which themselves are not embodied mathematics, and requires working with abstractions right away.

Our goal is to close the gap between the two ends of the spectrum, retaining the embodied, iconic elements of explainable frameworks, and yet allowing the user to compose and explore algebraic expressions. Existing works such as Bret Victor’s ladder of abstraction [47] or explorable explanations [44, 48] provide ways to see the link between different abstract representations, but do not construct symbolic mathematics from fundamental embodied mathematics natural to human beings. They rather use symbolic mathematics to explain symbolic mathematics [48], or primitives that are also further along the math spectrum in fig. 2, such as plots, coordinate systems, and visualizations.

We design and implement a framework that allows seamless construction and manipulation of mathematical expressions from iconic elements (e.g., sketches), giving users the power to assign mathematical semantics to personalized, whimsically sketched pieces. The scope of our work is limited to standard symbolic algebra, but we aim to lay the groundwork for what an embodied mathematical representation promises in the future for other types of mathematics.

The challenge in designing such a framework is the mapping between sketching, iconic elements, symbol algebra, and functions.

Our first contribution is a brief discussion connecting abstract math, sketching [8], and cognitive science, and a formative study on scientific diagrams. We formulate the motivation and design goals from this discussion. Our second contribution is a design framework that describes the above mapping. The framework consists of three unique brushes (figs. 6 to 8) and two key design ideas: fused representations (fig. 9) and abstraction layers (fig. 12). We instantiate the design framework by implementing an example sketch interface called Noyon, demonstrate resulting artifacts, and evaluate its affordances. The sketch interface and evaluation are our third contribution.

Using Noyon, users can sketch iconic elements (e.g., stick figures) to represent a visual category, create an unordered list of such elements (fig. 1 (brushes)), and apply simple mathematical functions (e.g., addition) to the lists and attributes of iconic elements by the function brush (fig. 1 (street containing the house lists)). Fused representation capabilities allow users to take advantage of the fluid and ambiguous nature of sketching, yet assigning concrete mathematical semantics to their sketches. Noyon also facilitates fluid transition between the symbolic and iconic representations (fig. 1 (abstraction ladder)), thus enabling users to navigate through the ladder of abstraction seamlessly. We evaluate the understandability of the sketch interface by playtesting sessions with children, and by adapting the Common Core math curriculum through designing exercises in Noyon. We also discuss the implications of our design framework for future math UIs for storytelling, education, and exploration.

## 2 RELATED WORKS

### 2.1 Mathematical Programming and Algebra Systems

Traditional Computer Algebra Systems (CAS) have relied on symbolic algebra so far. Some algebra systems [33, 42] only allow users to work with variables and expressions. Other systems [9, 14, 31, 32, 35] provide opportunities to blend programming with symbolic algebra. In all of these systems, the key focus is symbol manipulation. The visual output of CAS are important in comprehending the symbolic algebra and expressions. Many of these systems provide a notebook interface for composing text and producing plots and diagrams [14, 21, 30, 32], in the style of “computational essay” [51], where primitives like text, diagrams, and plots are used in a highly linear fashion, which may hinder dynamic composition and understanding of mathematical concepts. To ease the use of the above UIs, where the input devices are always keyboards and the output is on a screen, some other designs have been proposed previously, such as interactive touch interfaces [11, 56] and augmented reality to manipulate symbols [17]. Works by Bret Victor [46, 48] provide opportunities to the user to explore mathematical documents. In all of these systems, the focus is still on symbolic expressions.

### 2.2 Diagrams and Visual Programming

Visual programming tools like [13, 15, 37, 38] allow users to create dynamic diagrams and art using blocks or simple text syntax, while other methods such as [1, 40] provide users a hybrid text + diagram based grammar to create interactive diagrams. In all such tools,

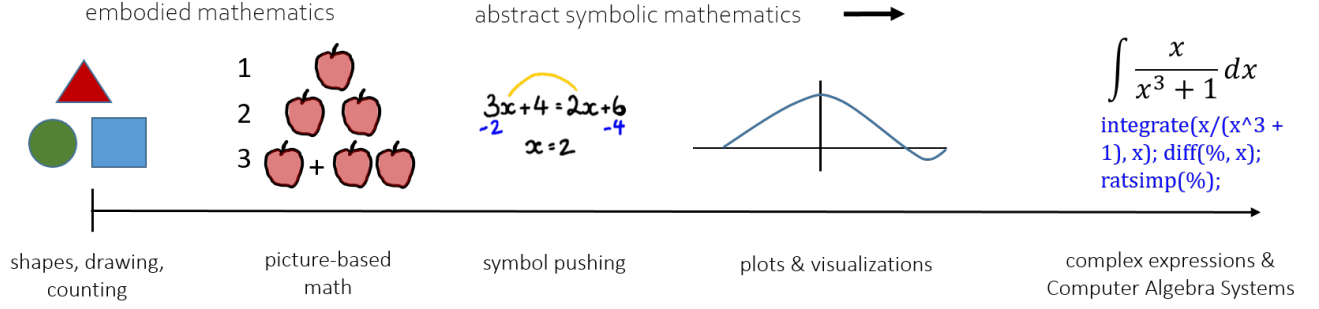


Figure 2: A spectrum of mathematical activities.

the programmable properties of the visual and iconic elements are usually manipulated using symbolic expressions. Visual programming languages [2] rely on a node-edge based paradigm to create animations and signal processing chains. A survey of categories of visual programming and direct manipulation can be found in [34]. Noyon differs from the described categories of visual programming: we aim for constructing, instead of directly writing, symbolic expressions from drawn iconic elements.

## 2.3 Sketching tools in HCI

**2.3.1 Sketching explanatory, dynamic diagrams.** Prior work explored sketch-based interface for the creation of dynamic diagrams, including interactive illustrations [18], dynamic fluid systems [57], physical phenomena [41], and interface designs [23, 27]. These tools often employ a graph model to represent the dynamics of a system [18]. Sketching has also been explored to create expressive and custom data visualizations [25, 29, 54]. Other visualization systems explored sketching as an activity for sense-making [39] and data exploration [26, 49]. Our aim is to build a medium where the sketching, thinking, and building up of symbolic algebra happens seamlessly through iconic elements and functional mappings.

**2.3.2 Mathematical Sketching.** Mathematical sketching tools [5, 11, 24] facilitate the formulation of mathematical expressions using sketching, where the hand-drawn diagrams animate in response to parameter changes in the written formulas. In contrast to prior systems, the construction of mathematics starts from iconic visual elements, thus enabling users to think through visual and mental models [16, 53].

## 2.4 Design Space

In the design space of generativeness and interactivity (fig. 3), our design framework and system could be seen as generating symbolic expressions through visual/embodied interactions. Penrose [55] is a system that generates visual diagrams from symbolic expressions. Mathematical sketching works like Mathpad [24] is at the middle of abstract and visual math, but the emphasis is usually on symbols. Works like "explorable explanations" [44] are customized, non-generative in nature, and children's math learning apps (such as Zoombinis) are visual but heavily customized. Additionally, our work provides ladder of abstraction [47] for exploration.

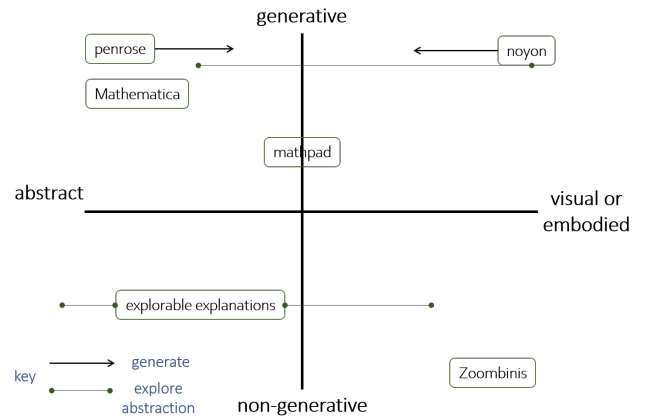


Figure 3: Situating our system, Noyon, in the design space of generativeness and visual/embodied interactions.

## 3 MOTIVATION AND DESIGN GOALS

In this section, we discuss how sketching and embodied mathematics theories can inspire designs for fluid and natural computer algebra systems used for explanatory and exploratory mathematics. We also present an informal formative study that identifies three characteristics of scientific diagrams. Based on the discussion of these ideas, we recommend a few design goals to bridge the spectrum between embodied and abstract mathematics using sketching.

### 3.1 Embodied Mathematics

The embodied mathematics theory [22] posits that our innate mathematical capabilities are limited to counting, categorizing and grouping. The authors proposed that there are four *grounding metaphors* (or the "4G"s as the authors call them) that help us understand and extend to higher level arithmetic and math operators, which are based on our innate capabilities and require little instruction. The 4Gs are, a) *Object Collection*: categorizing and grouping objects, (b) *Object Construction*: understanding parts and whole of an object, and composing different objects from different parts, (c) *Measuring Stick*: denoting an amount/cardinality by the dimension of an object, and addition and subtraction as increasing/decreasing the

length, and (d) *Motion along a Path*: similar to the *Measuring Stick*, but using the travelled length from the path origin as the unit. Their book [22] further describes how advanced mathematical concepts are created based on these metaphors.

Based on the *grounding metaphors*, there are *linking metaphors* that then establish connections to even higher level concepts (metaphors such as a number as a point on a line, geometry figures as algebraic equations). In fact, the authors demonstrate that abstract mathematics consists of layers and layers of such metaphors that were established over many centuries.

We take this philosophy as our design guide. Our aim is to leverage the four *grounding metaphors* to design interactions for computer algebra. It should also be noted that a lot of the traditional abstract mathematics that we learn are *linking metaphors*, which require explicit instructions and long training.

Central to embodied mathematics is the cognitive container schema [22]. Ordinary, everyday cognition (such as simple spatial analysis) and mathematical cognition are often derived from similar embodied experiences. Figure 5(b) shows an example. Visual inspection of an object containing other objects gives us a sense of containers and hierarchies. Venn diagrams are comprehensible precisely because of the container schemas.

### 3.2 Deriving Interactions from the "4G"s

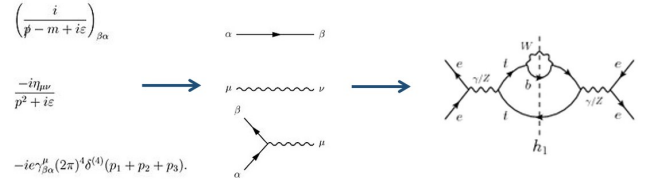
There are some intuitive mathematical interactions that can be designed from the *grounding metaphors* of embodied mathematics, when an interactive sketch interface is our primary medium to embody mathematical concepts. The interactions are:

- (A) **Categorization:** Sketched forms can be categorized by either the human or machine before forming groups.
- (B) **Grouping Objects:** Following the *Object Collection* metaphor, one could imagine designing interactions to draw and manipulate groups of objects.
- (C) **Composing:** Following the *Object Construction* metaphor, composition of groups could lead to hierarchies of groups, and compositions from different parts could form whole new mathematical objects. Applying the cognitive container schema here means that hierarchies could be defined by the container metaphor (fig. 5(b)).
- (D) **Graphical Transforms:** The last two Gs (*Measuring Stick* and *Motion along a Path*) could be reproduced in a sketch interface as graphical transformation capabilities. The ability to define parameterized paths, translations, scaling etc. in a dynamic sketch interface could be utilized, using mathematical compositions to drive iconic object transformations.

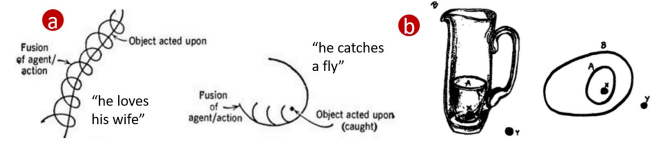
At this point, the sketched objects, groups, hierarchies, and compositions are not precisely defined. We will elaborate them in the design framework section.

### 3.3 Diagrams in Mathematical Modeling

After an informal investigation on 20 diagrams (see supplementary files) that are used in different kinds of mathematical models and applied mathematics, we have identified three main characteristics relevant for our design. We demonstrate them here using a famous diagram in particle physics: Feynman diagrams (Figure 4).



**Figure 4: Feynman Diagrams describe how different particles interact with each other. Elements of symbolic representations are denoted by vertices and edges with certain properties, and diagram rules dictate how to compose a graph.**



**Figure 5: Two relevant ideas from cognitive science. (a) Fused representations found in sketch studies (Symbol Formation, 1963) [50]. (b) Cognitive Container schemas (Where Mathematics Comes From, 2000) [22].**

*Ladder of abstraction.* The gradual layers of abstraction is common to reduce symbol clutter and symbol manipulation by introducing iconic portrayal of symbolic expressions. In fig. 4, for example, there are two layers of abstractions. Denoting smaller expressions using some diagram elements, one may use these elements and a set of rules to compose bigger diagrams. Such ladder of abstraction [47] makes it easy to communicate, brainstorm, and even discover new rules [36] in the diagram space, without resorting to pages of equations.

*Ease of moving along the ladder.* Often, moving along the abstractions is a computationally expensive process, and making them interactive poses further challenges. Additionally, most of the diagrams we have studied are uni-directional in their nature of abstraction, because they are mostly used for interpretation purpose. In Feynman diagrams, no algorithms exists that automatically propagate the changes in visual composition to the equation space.

*Compositionality.* Some diagrams (e.g., Feynman, Penrose) have specific rules, so a composition of more complex structures are possible without resorting to symbolic manipulation. It is important to be able to define a grammar for such interactions, informed by the underlying mathematics, to truly utilize the ladder of abstractions. Interestingly, compositionality is a key concept in the *Object Construction* metaphor in embodied mathematics.

### 3.4 Fused Representations in Symbolic Activity

Prior studies show that people often use visual forms to depict actions on objects when sketching; these are called fused representations [50] (fig. 5 (a)). According to the drawer's personal interpretation, actions are given embodied forms by assigning them



to drawings. We propose that one way to fuse mathematical actions with drawn objects is to implement the cognitive container schema (fig. 5(b)). For example, instead of drawing a generic shaped Venn diagram to denote the domain and relationships between the pitcher and water (in fig. 5(b)), a sketch interface could provide the option to just draw such a relation in the shape of the pitcher containing water.

### 3.5 Symbol Grounding

In mathematics, we usually need to ground an expression in some real world experience and properties of objects to create meaning out of the symbols. For example, once we ground the symbols  $x$  and  $y$  to denote length of two sides of a triangle, we can create more complex expressions like  $x^2 + y^2$  without the need to define new grounding for this expression. However, a challenge for mathematical symbol grounding is that it is dynamic. ‘ $X$ ’ could mean number of apples or someone’s heart rate in an algebraic expression or equation, depending on the problem the expression is modeling. This hints at designs that construct the grounding starting from iconic descriptions instead of symbols, and retain such description when composing more complex expressions.

### 3.6 Summary and Design Goals

In light of the discussion and investigation in this section, we formulate the following design goals.

- D1** We want to preserve the *fluid* and *ambiguous* [8, 10, 19] nature of sketching to construct precise algebraic expressions and mathematical models. This can be achieved by utilizing the features and interactions defined in section 3.2, and using fused representations (section 3.4) to assign mathematical semantics to sketched objects.
- D2** Facilitate a *bottom-up approach* in constructing mathematics, inspired by symbol grounding. Let the users construct symbolic mathematics by sketching and embodied interactions to bridge the two ends of the math spectrum. Layers of iconic metaphors can be constructed based on interactions inspired by the *grounding metaphors* (“4G”s), which may generate algebraic expressions in the higher abstraction layers. This is essentially our definition of *embodied algebra*.
- D3** Facilitate fluid *compositional*ity to enable fast and easy compositions of iconic elements.
- D4** Implement *ladder of abstraction* to enable fluid transition between symbolic and iconic representations. This is an attempt to create mathematics that provides *ease of movement* along the ladder, and clarifies the links between the iconic, metaphorical representations to algebraic expressions.

## 4 DESIGN FOR EMBODIED ALGEBRA

The universe of mathematics and mathematical modeling is vast. We demonstrate the first step towards a design framework that can use visual metaphors as embodied representations of abstract mathematics. We start with simple mathematical operations and sets of objects, and demonstrate their power by combination with sketching, abstraction layers, and lasso-based list creation and function mapping. A sketch interface named Noyon is created as one possible implementation of the framework.

### 4.1 Elements of the Framework

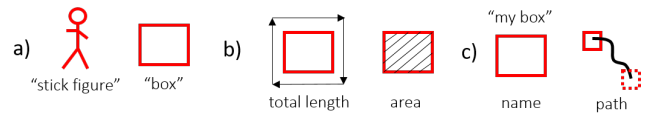
In this section, we describe the relevant definitions and elements that are used in the design framework.

**4.1.1 Variables and Symbolic Expressions.** A variable is a numerical quantity on which a final modeling outcome relies on. A symbolic expression is a collection of variables on which mathematical operations such as addition, subtraction, multiplication, division, exponentiation etc. have been applied.

**4.1.2 Mathematical Model.** A mathematical model ‘ $y$ ’ is an algebraic expression  $f(\vec{x})$  which consists of variables  $\vec{x}$  [6]. In this definition, the variable quantities are represented under a vector  $\vec{x}$ . In reality, they may not be named ‘ $x$ ’ with vector indices. They can take on any name, similar to a variable in a programming language. There are many types of methods that predict or infer the expression  $f(\vec{x})$ , e.g., differential equations, statistical modeling, numerical optimization etc. Here, we focus on constructing an expression or a model from the embodied interactions defined in the previous section. An algebraic expression can be deemed a model of a phenomenon as long as the variable values are changeable to predict the outcome.

**Scope of Algebra:** There are many types of algebra in advanced mathematics (e.g., Lie algebra, Boolean algebra, homologic algebra etc.). We focus on computer algebra implemented in most CAS software, which covers algebra usually taught up to college level mathematics and used in most applications (outside advanced mathematics).

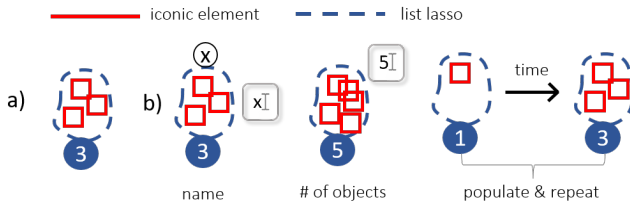
**4.1.3 Iconic Elements.** Iconic elements are central to our design, representing a unit element of a model. They are composed from a stroke drawn in Noyon. Imported images are also treated as iconic elements. Multiple strokes or images can also be grouped together to create one iconic element (fig. 6(a)). They can be moved with touch or by a user-defined translation path. Attributes of iconic elements (fig. 6(b, c)) include name (each element can be named by the user or assigned a default unique name), length (total length of strokes), area (area of the polygon that encompasses all the strokes in an element), and user-definable graphical transformations, such as a path that can be used to translate the element.



**Figure 6: Examples of iconic element categories (a), and attributes that are gettable (b) and settable (c).**

**Categories of Iconic Elements:** Each iconic element has an assigned category based on its visual resemblance to a set of objects (for example, arrow, triangle, square, and other user-defined categories). An iconic element that is not recognized is assigned a default category. Such categorization helps us create a visual definition of any variable that will contain these iconic elements, and ensures that the ambiguous and imprecise sketched forms are assigned a concrete category to help later construction of variables and expressions in a bottom-up manner.

**4.1.4 List.** The second structure in our design is an unordered list of iconic elements, which keeps a count of how many elements are within (fig. 7). The data type of each iconic element in the list is its assigned category. The list can be drawn in any shape, and all encompassing iconic elements become its members. Because the membership criteria is flexible to any drawn shape, the user can take advantage of their visual preference to select their domain of choice. The list has one output attribute: the total number of contained iconic elements, and one name attribute definable by the user. A list can also contain other lists and iconic elements to count their total in a nested fashion. The counting operation is therefore a summation function.

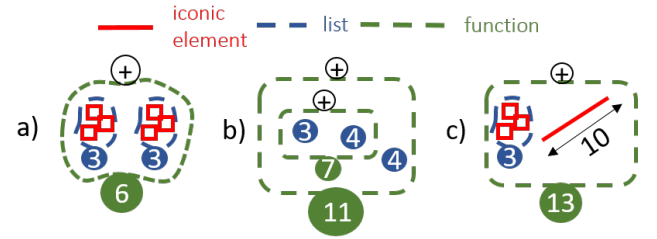


**Figure 7: (a) An unordered list of iconic elements (membership defined by inclusion in the “lasso”). (b) Some settable properties of lists created in this way.**

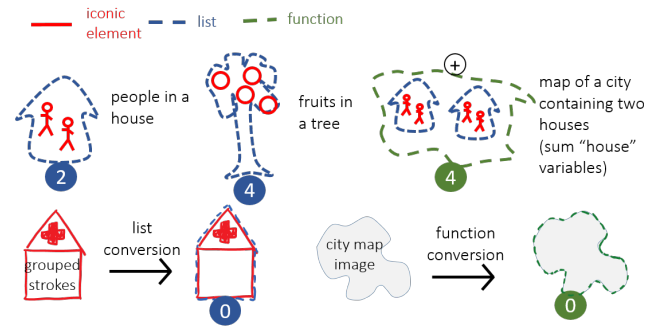
**4.1.5 Function.** Functions allow a mathematical operation, such as addition, multiplication, etc., to be mapped on a domain present in the scene. Like lists, they can be drawn flexibly and in any shape, thus facilitating fluid domain mapping. The domain can be unordered lists, iconic elements, or other functions (fig. 8(b, c)). A function can take all three kinds as its arguments. For iconic element arguments, the function acts on the (user-selectable) length or area of the element. For list arguments, the function acts on the total count (cardinality) of the list. Operations like sum and product ( $\sum_{i=1}^a f_i$  and  $\prod_{i=1}^a f_i$ , where  $f_i$  is the output attribute of the  $i$ 'th argument) can be applied seamlessly on any number of arguments, where the user determines the number of arguments (' $a$ ' in the formulas) and the specific arguments by the shape of the function. For operations that require ordered operands, such as subtraction, division, etc., arguments within the function can be ordered by the user during function specification. The numerical output attribute of a function lasso is the evaluated value of the operation.

**4.1.6 Fused Representation.** Mathematical structures like lists and functions can be drawn in any shape (fig. 9 top). A group of strokes (representing iconic elements) or an imported image can also be converted to a list or function structure, where the convex hull of the iconic element(s) represents the boundary of the list or function (fig. 9 bottom). These *fused representations* allow the user to draw and compose mathematical expressions in visual forms (“fusing” abstract math relations and visual forms). For example, one can represent math expressions visually for people in a house, fruits in a tree, or a “city” summation function that adds people in the house.

**Importance:** Fused representations allow users to interpret and assign their own mathematical meaning to any visual form they

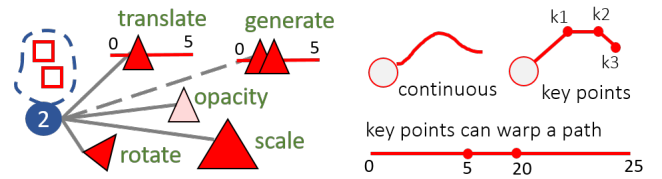


**Figure 8: (a) A summation function with two lists as arguments, (b) hierarchical nature of drawing/mapping functions, outer function taking in two arguments (another function and a list), (c) a function acts on a property (length) of iconic elements and cardinality of lists.**



**Figure 9: Fused Representations. (Top) Lists and functions in the shape of visually relatable concepts. (Bottom) Single and group of iconic elements can be converted to lists and functions.**

sketch. This also helps Noyon deduce operator semantics from ambiguous, whimsical sketches made by the user, as long as the user marks their intentions for what each sketch (or a group of sketches) might mean mathematically.



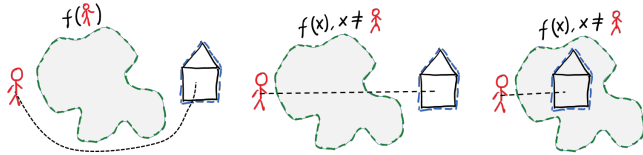
**Figure 10: Graphical transformations for dynamic illustrations. This example shows how the count of a list can dynamically drive the movements of iconic elements along a path (or apply geometric transforms) (left), in continuous or discrete steps (right).**

**4.1.7 Dynamic Illustrations.** A node-edge type relationship can be created from a list or a function to drive (user-defined) graphical

transformations of any iconic element (fig. 10). Possible transformations are: (a) translation over a user-defined path, (b) particle generation (the value of the driving list/function dictates how many), in which each particle is a copy of a template particle with a path like (a), but other parameters such as movement stochasticity and looping can be added, additional attributes including (c) opacity, (d) rotation, and (e) scale. Each edge can also have a conditional expression assigned to it to enable/disable it. These features facilitate the creation of dynamic illustrations like [18], and increases the expressiveness of any sketch in the spirit of design goal D1.

The paths for (a) and (b) can be defined as a continuous stroke, or a discrete stroke with editable key points. Thus key points can warp a path, making iconic elements move more or less for the same range of list or function output values. These path features also facilitate membership definitions in lists and functions.

**4.1.8 Defining Membership to Lists and Functions.** There are four ways to change values of lists and functions. Essentially, they allow us to formulate and simulate algebraic expressions by changing the children variable and function values (denoted by the "compose" arrows in the framework). (1) Move sub-compositions in or out of parent compositions, e.g., moving iconic elements to lists or functions, or moving lists to functions. (2) Populate lists or functions with template compositions or iconic elements (essentially, "going down" the abstraction ladder). (3) Define a path for an iconic element to move (fig. 10), getting included in or excluded from any list or function that the element enters or leaves during its movement. (4) Use particle generation (fig. 10).



**Figure 11: Category filters.** (Left) To avoid being included in the domain of a generic function  $f$  (city map) while intending to be included in a list (house), the only way is to draw a path around the function. (Middle) introducing a *category filter* in the function to exclude the stick figure allows one to let it pass through without being included in  $f$ . (Right) Membership is based on immediate hierarchy. When the figure is inside the house, it will be registered by the house, even though the house is part of the function.

Membership programming by methods (3) and (4) are facilitated by *category filters*. As shown in fig. 11, filtering categories is a useful feature to control the domains of functions and lists (essential for mathematical modeling, and a manual task that is usually tracked by the CAS programmer over many lines of symbolic expressions), and provides flexibility to the user to draw and define animations naturally, without worrying about what falls in the path of sketching.

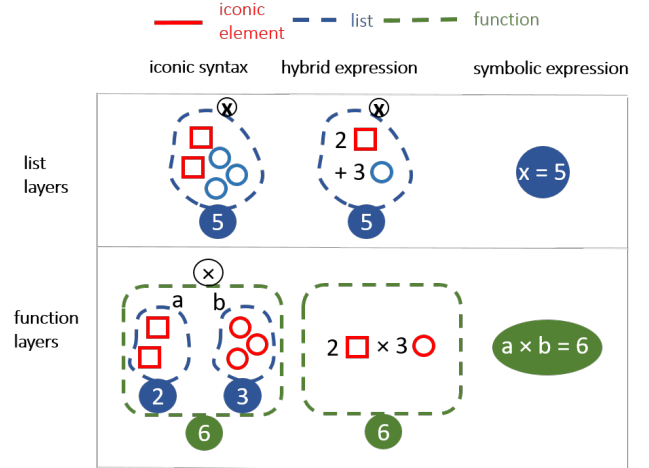
## 4.2 Deriving Algebra from Sketches

Given the design components so far, we can now formulate a way to create algebraic expressions out of them. This relies on our design of abstraction layers.

**4.2.1 Abstraction Layers.** Both list and function structures have three layers of abstraction: iconic, hybrid, and symbolic (fig. 12). Any composition (made up from lists and functions) can be explored in these three layers.

The *iconic* layer shows the entire structure of a user-sketched composition. The *symbolic* layer shows the text-based expression arising out of the iconic composition. Also note that the symbolic layer folds up the iconic description of its children, and locks membership changing capabilities of children elements by doing so.

The *hybrid* layer is an intermediate between iconic and symbolic expressions, an aggregate of constituent children's iconic element categories that show how different categories are consolidated in the final symbolic expression. This provides a visual dimensional analysis of the algebraic expression. For example, if we use the category 'box' to denote a person, and a 'circle' to denote unit of time like hour, then fig. 12 (bottom) dimension layer shows a unit of  $person \cdot hour$  ( $6 person \cdot hour$ ). With user-defined iconic element categories, one can imagine building up visual dimension expressions in this way, for example,  $meters/second$ ,  $person \cdot hour$ , etc.

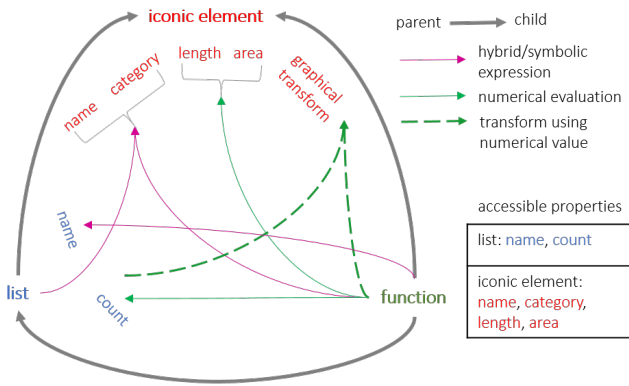


**Figure 12: Abstraction layers for lists and functions.**

**4.2.2 Building up Expressions from Names and Categories.** Every computer algebra system needs to have a coercion model [7] in their design, which displays a natural output when different types of mathematical objects are combined in an expression. In Noyon, the visible output in each layer of a parent list or function follow a coercion model that is outlined in fig. 13. The coercion model is represented in the design framework (fig. 16) by purple arrows.

To create a symbolic expression for a function, the 'name' attributes of children iconic elements and lists are accessed. For a dimensional expression, the categories of iconic elements are accessed. Since lists do not have any specific category in our design,



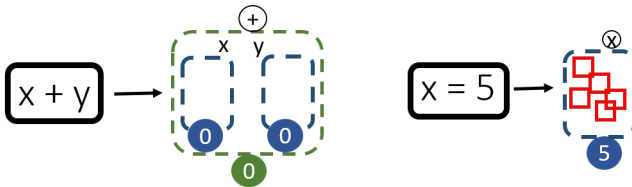


**Figure 13: Noyon coercion model.** Functions and lists can access specific attributes of children iconic elements or lists in order to create numerical, hybrid, or symbolic expressions (three abstraction layers).

dimensional expressions always show an aggregate unit expression based on all children iconic elements, aggregating categorical expressions recursively. Numerical evaluations are calculated by children’s counts (for lists), length or area (for iconic elements), or evaluated values (for functions).

*Reusability of a Composition.* A composition can be reused in the sense that it can be applied to other iconic elements, lists, and functions distributed in a similar fashion as the current arguments, by moving or copying the parent composition as a template and placing it on top of the distributed children composition. It is the sketch equivalent of a reusable function in programming languages.

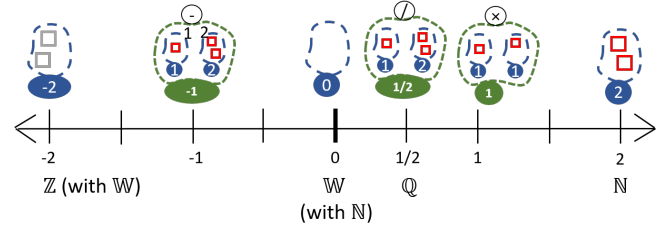
**4.2.3 Bidirectionality of Abstraction.** Even though we emphasize *bottom-up* construction in our design, it is possible to “go down” along the ladder of abstraction in Noyon from symbolic expressions to sketched composition templates. For plain symbolic expressions, an iconic layer template can be generated by Noyon that can be pre-filled (fig. 14 (right)) or left to be filled up by the user or to be applied on a particular distribution/composition (fig. 14 (left)).



**Figure 14: Bidirectionality of abstractions: going down the ladder of abstraction from symbolic to iconic layer.**

**4.2.4 Number Domains.** All common number domains are covered (fig. 15). They can all be constructed (or can be generated, fig. 14) through sketched compositions (natural numbers  $\mathbb{N}$ , whole numbers  $\mathbb{W}$ , integers  $\mathbb{Z}$ , rational and fraction numbers  $\mathbb{Q}$ ). Although negative numbers, other than construction by functions, can be generated as lists containing dummy iconic elements representing

negative entities. We do not cover advanced types such as complex numbers ( $\mathbb{C}$ ), and leave the designing of embodied/iconic representations of such number domains as future work.



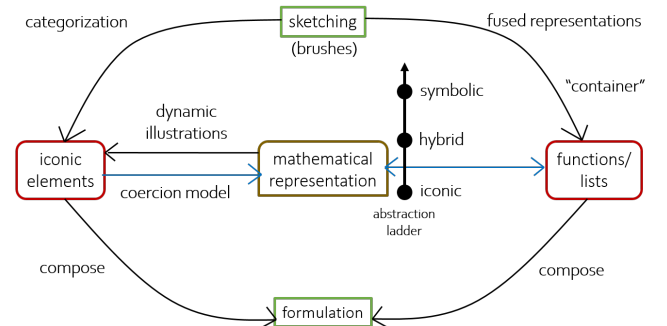
**Figure 15: Number domains covered by the framework.**

### 4.3 Design Framework

The design framework illustrated in fig. 16 essentially summarizes all the mappings between different activities (sketching, and formulating algebraic expressions) and the sketched forms (iconic elements, lists, and functions). Forms such as lists and functions define mathematical relationships and domains. Different functions and expressions can be composed by intuitive interactions. Dynamic illustrations, in conjunction with gestures and path animation to compose different formulations, can be used to create graphical transforms. A coercion model (usually prevalent in computer algebra systems, tweaked for our purpose) can be used to generate algebraic expressions for any composition’s symbolic abstraction layers.

### 4.4 Design Summary

Sketched compositions (iconic elements and categorization (fig. 6), lists (fig. 7), functions (fig. 8)), dynamic illustrations (fig. 10), and fused representations (fig. 9) retain our goal D1, to preserve the flexible and fluid nature of sketching and yet accommodate ambiguity in doing precise mathematics. The hierarchical and compositional nature of each of the primitives, and tools that help enable fast



**Figure 16: Noyon design framework at a glance.** It maps and connects the activities (sketching, and formulating algebraic expressions) and main elements (iconic elements, lists, and functions) by different interactions and ideas from embodied mathematics.

compositions (such as defining membership to compositions to simulate expressions (fig. 11)) aim to solve D3. The abstraction layers mechanism (fig. 12, fig. 14) satisfies D4. Finally, the embodied interactions (grouping, moving, lassoing, drawing etc.) inherent in our design, the coercion model (fig. 13) to ground symbols in drawn categories, and fused representations (fig. 9) to convert personalized, whimsical drawings to concrete mathematical structures satisfy D2 (symbol grounding and embodied algebra).

*Connections to grounding metaphors.* The interactions and elements in the design framework are inspired by the proposed interactions in section 3.2. Categorization of iconic elements implements section 3.2(A). Lists and function structures are motivated by the *Object Collection* metaphor (section 3.2(B)). Composing to formulate different expressions is motivated by the *Object Construction* metaphor (section 3.2(C)), along with the hierarchical nature of lists and functions resembling the container metaphor. Dynamic Illustration capabilities implements the proposed graphical transforms (section 3.2(D)).

## 5 NOYON: INTERFACE

Based on the design components in the previous section, in this section we describe the user interface and workflow of Noyon. We implemented all features from the design framework, in order to explore its affordances by ourselves. However, depending on the target users and their age, form factor (tablet or smartphone), and curriculum, the features and scope of functionalities could be different for another implementation.

The UI of Noyon consists of a drawing canvas and a tool widget (fig. 17).

### 5.1 General Workflow

*Workflow.* A user draws iconic elements using the predictive or freehand stroke pens. Images can also be loaded as iconic elements and assigned a category. The list brush (fig. 17) allows the user to draw lassos around these iconic elements to create unordered lists. A function can be drawn in a lasso style encompassing the lists and iconic elements to create algebraic expressions. Each primitive can be dragged using pen or touch. Children compositions are locked with parent functions/lists so they move with the parents. Any primitive can be dragged out to undo its membership in a composition. An example workflow is shown in fig. 18, following the summation problem shown in fig. 1.

*Strokes.* A pencil mode allows the users to draw plain strokes and labels that are not meant to be registered by the list/function lassos or touch interactions. There are two types of pens for drawing iconic elements. A user can draw a freehand stroke to create an iconic element. They can also choose to use the predictive stroke, which predicts what a collection of strokes might represent, and assigns a category to each iconic element.

*Visual Details.* At any time, a user may decide to turn off math details (such as numbers, expressions, lassos etc.) by a visual detail dropdown checkbox. A user may want to turn off math details for emphasizing the fused representation: each sketch's embodied user-imagined meaning and their intended actions.

*Defining Paths.* Using the path brush (fig. 10), a user can hold on an iconic element and draw their preferred path for that element

using desired number of strokes. Each key point or end point of a path can be tapped to edit its range value.

*Linking list and function outputs to drive iconic element transformations.* The graph mode allows users to link a source (list or function) to a target (iconic element) with two variants: one is for driving an element for graphical transformation, another for particle generation. The edge created from such an interaction can be long-tapped to bring up menus to adjust details such as condition expression to enable/disable the edge, properties of particle generation etc.

*Copying.* A copy brush (fig. 18 (b)) allows the user to hold the brush on top of the desired primitive and draw in any direction to distribute copies accordingly. Edges and all other associated properties are copied automatically.

*Iconic Element Conversion.* Under the list and function conversion modes, a user can convert any group of strokes or images into a list or function primitive. The iconic elements are deleted in the process, but the strokes are retained with the list/function (fig. 18 (d)). This is a useful operation for fused representations. It allows users to sketch any iconic element, and then turn it into a mathematically actionable entity.

### 5.2 Menu Operations

Each iconic element, list, and function has an accompanying menu that can be brought up by short tapping on them (Figure 19).

It is possible to set the number of iconic elements from a list menu (fig. 19b) without having to create each and every element; the first element in the list is copied over by our system. There is an order selection menu button (fig. 19c) that allows a user to tap on children elements to assign order, which is particularly helpful when a function requires ordered arguments. The operator textbox in fig. 19c lets the user define a function's operations on its children. Finally, category filters (fig. 11) can be accessed from list and function's context menus. An abstraction slider lets a user move back and forth between iconic, hybrid, and symbolic layer seamlessly for lists or functions.

*5.2.1 Operations on Hierarchical Structures.* As hierarchical compositions are drawn/constructed iteratively, some tools are provided to operate on them. A Flatten operation dissolves a composition and releases the children iconic elements recursively, whereas the Unlink operation separates the composition and children iconic elements. Unlink is particularly useful for reusable function templates.

## 6 IMPLEMENTATION

Noyon's prototype was built in Unity. It runs an open source computer algebra system in the background [3]. The symbolic and hybrid layers of abstraction use standard algebra algorithms such as rationalizing and standardizing symbolic expressions, so that when aggregating an expression like  $x \cdot x \dots \cdot x$  in a composition, it is simplified and shown as  $x^n$ . Stroke prediction is done using the q-dollar stroke recognition algorithm [43].

Unlike sequential evaluations typical in CAS and programming languages, each composition in Noyon is evaluated independently (fig. 21), and their outputs can be consolidated by using them together in other compositions. Compositions proposed in our design

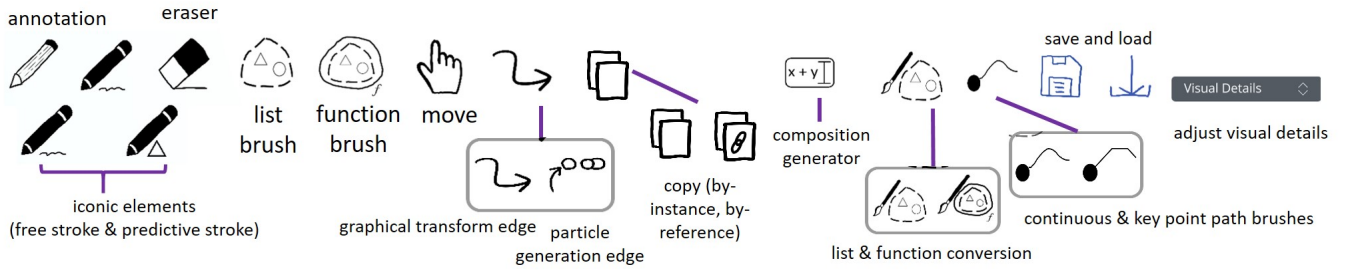


Figure 17: Noyon UI widgets.

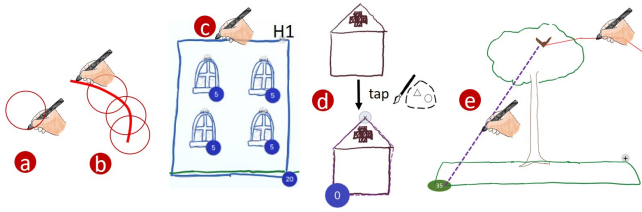


Figure 18: Working in Noyon. (a) Iconic elements can be drawn using their respective brushes. (b) Iconic elements (or lists and functions) can be copied in any desired pattern. (c) Lists or functions can be drawn, or (d) created from iconic elements by element conversion. (e) Dynamic illustrations are achieved by connecting an output to an iconic element through graphical transform edge, and a path brush allows us to define translation paths for iconic elements. Other graphical transforms are possible through the menu options.

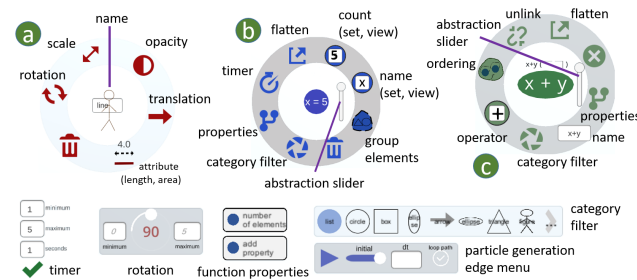


Figure 19: (Top) menu operations for iconic element, list, and function. (Bottom) examples of sub-menus for setting properties.

is directly convertible to usual prefix notations and nested lists used by CAS to store expressions [28].

## 7 EVALUATION

To evaluate the breadth of the compositions and interactions afforded by our design, we adapted the US Common Core mathematics curriculum syllabus [4] and designed example exercises for each prescribed section in it. Additionally, a few interactive examples

were devised and explored by us to further understand the affordances. Playtesting studies were carried out with 28 children in 3 schools in Bangladesh and also with 15 children in the USA.

### 7.1 Common Core Curriculum Adaptation

The Common Core curriculum often asks students to employ pen and paper to work on visual examples. Our design framework provides an alternative to many example exercises offered in the Common Core guidelines, providing an opportunity to take advantage of the sketch interface and associated interactivity. Appendix A provides our methodology and some common examples for each theme in the curriculum from Kindergarten to middle School (8th grade). Overall, arithmetic and algebraic sections were well covered by our interface (100% coverage for the sections Counting and Cardinality (CC) and Operations and Algebraic Thinking (OA)). Around 80% of the sections Measurement and Data (MD) and Numbers in Base Ten (NBT) could be covered. However, our design fell short in sections such as Statistics and Probability (SP), where the curriculum explicitly asks to use representations that are not currently supported (e.g., number lines).

### 7.2 Playtesting with Children

Asian math education often differs from the US with the introduction of Common Core. The Bangladeshi curriculum often puts emphasis on symbolic expression based mathematics in elementary schools [12]. The goal of this playtesting study was to observe how Bangladeshi students work with some basic math afforded by the sketch interface, and later compare the findings with studies done in USA. The two education systems rely on symbolic expressions to different degrees, hence the comparison would be particularly suitable to test the effectiveness of our design. The study was not designed to be a summative study but rather the goal was to do a qualitative, formative study to explore how students from different training and backgrounds use the above brushes to solve and create stories with simple arithmetic.

**7.2.1 Participants.** 28 children were recruited in three schools (S1, S2, S3) in Bangladesh. S1 was a religious school where sketching and painting are discouraged, and students were not exposed to any advanced arithmetic (9 children, all boys, ages 9 - 12). S2 was a public school in a remote village named Feni (11 children, 5 girls, ages 9 - 11). S3 was a private, English-medium school in the capital Dhaka (8 children, 5 girls, ages 9 - 11). In the USA, 15 children were

recruited in Boston, Massachusetts (6 girls, ages 8 - 10). Students in S1 and S2 were from disadvantaged backgrounds, and did not have access to tablets or smartphones beforehand. S3 and USA children were very familiar with tablets and smartphones, and educational apps and games. For each location, we report results from the first day sessions. The sessions were generally around 2 hours long, and each session took place in the respective locations of S1, S2, and S3. In the US, the session was conducted in our lab.

**7.2.2 Tasks.** Before each session, a basic training familiarizing the students with the three brushes and illustration capabilities were provided by means of some examples. In S1 and S2, each tablet was shared by two children and they discussed and worked in teams. All four groups were given three tasks: A) Draw iconic elements and lists, and move iconic elements in/out of lists to count. B) Simple addition and subtraction with lists and functions. C) Freeform storytelling with addition and subtraction operators, and dynamic illustrations. To do the tasks A and B, students would have to learn and use the iconic elements, list, copy, path brushes, and pan mode. We measured success qualitatively, by analyzing how they were able to use these sketching concepts to solve and create simple math problems.

**7.2.3 Results and Discussion.** All participants could finish all the tasks, although the timing for each task varied widely. For tasks A and B, on average, S1 students took 1 hour, S2 students 36 minutes, S3 students 21 minutes, and US students took 24 minutes. Task C was administered as an hour-long session in each case, and everyone had at least one story completed within that time. Our observation data consisted of video recording, hand-written notes, and pre/post-session interviews. Multiple observers independently coded and discussed the data to organize it into a few themes described in this section.

**Participant Feedback.** Reactions varied across the different sites. S1 and S2 students, with no initial experience of such devices or apps, took more time to familiarize themselves. However, many of them became very interested and spontaneous in the storytelling and presentation session. Students in S3 and US received the app positively. One child in the US session commented to her parent (at the end of the session) that she can "do this all day" (i.e., draw in Noyon). However, the novelty of a sketch interface as a medium for math startled a few. One child in the S3 session exclaimed that "this is not how you do addition!" while working with a function brush, and proceeded to write down the summation in the traditional number system in the sketch interface. However, she independently discovered that the results were the same, and later proceeded with the function structures to create her stories.

The US children's parents mentioned that their children were excited to attend the session and wanted to come back for more. One parent informed us that her daughter usually does not like to go to math tutoring session, but eagerly wanted to come back for more sessions with us. An instructor in S1 reported to us that his students asked for permission to use the only school computer to sketch with Noyon.

**Effect of Familiarity with Digital Tools.** Unlike S3 and US, S1 and S2 students were not familiar with digital tools. This was evident in their first interactions, initially requiring more help and explanation,

and also by the time taken to finish tasks A and B. Compared to S1 and S2 children, S3 and USA children also produced more freeform stories in the hour-long session. Some unique observations in S1 and S2 locations also informed our future work. Because many children in S1 and S2 were not familiar with stylus and digital drawing programs, they used their fingers to draw. Consequently, the pan and move tool, which looks like a hand sign, were repeatedly mistaken by many as a drawing brush. Secondly, English numbers used in lists and functions proved to be a challenge for a few students to interpret counts and outputs. S3 and USA students seemed more comfortable with the interface and in many cases already knew what the pan/move or copy tools did.

**Differences due to Symbolic Math Experience.** S1, S2, and S3's children seemed generally more exposed to symbolic math (number and symbol manipulation), as some expressed eagerness and asked specific questions about how interactive sketching could relate to mathematics. On the contrary, the US children's general response in a survey was that they were exposed to interactive apps for math before. Several students described doing math in the interface as "playing a game" or "making animation".

**Compositional Capabilities.** In the freeform storytelling session, many students utilized the compositional capabilities to create hierarchical structures. Hierarchical structures involving only lists (S1), and a combination of lists and functions (S2, S3, US) were also common. We often noticed the use of abstraction layers to fold and unfold such hierarchies.

**Expressing Mathematics through Stories.** Using Noyon, children created personalized stories and animations involving algebraic expressions. Fused representations were used by many to express the stories. A few examples follow. A child in S2 created a story of a bus moving people from his village to another, employing dynamic illustrations and both addition and subtraction operators. A girl in S3 created a story about ghosts in a house, employing dynamic illustrations and lists to move the ghost when people gather in a certain room. Another child in US created a story about doing his own grocery from a market, using fused representations to draw an addition function in the shape of a house, and using copies of lists named "apple" and "orange" that contained fixed number of cents (iconic elements drawn as cents), to calculate the total cost. Some of these examples are shown in appendix B.

### 7.3 Exploring Affordances

Abstract, symbolic mathematics mostly happens on pen and paper, or through computer algebra systems. Using the visual programming features of Noyon, we explored how novel activities can be afforded by embodied mathematics when it is implemented through a sketch interface. Figure 20 shows some results from our exploration.

## 8 DISCUSSION AND CONCLUSION

In this paper, we have described the motivation for why the embodied math philosophy should be implemented to reimagine abstract math. We presented a design framework, which uses three kinds of structures (iconic elements, lists, functions) and allows algebraic manipulation by sketching interactions inspired by the *grounding*

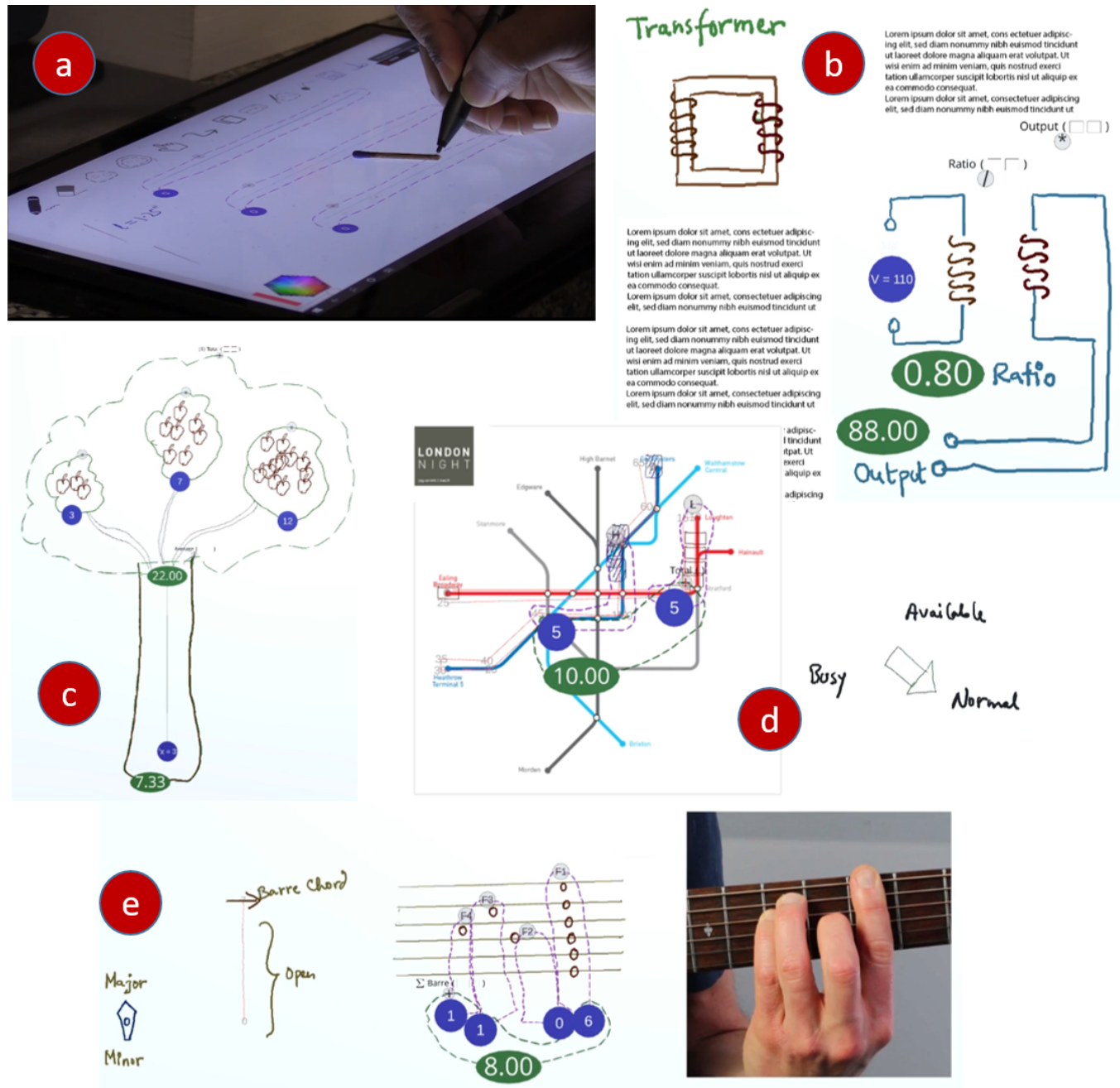


Figure 20: Please check the supplementary materials video for more explanation. (a) Calculating Pi using Buffon's Needle formula. Throw match sticks on top of Noyon and form Buffon's formula from the line crossings. (b) Extended version of "Explorable Explanation" document [44, 48]. By-reference copies of lists and functions allow one to draw in one portion of the document, and the rest of the document and formula update live, in all abstraction layers. Here, a list containing coil-turns (iconic elements in the top-left transformer coil) is copied into a visual formula for calculating step-down voltage of a transformer. Drawing more turns in the original transformer coil updates the by-reference copy, and hence, the final output value. (c) A dynamic average function that calculates the average number of apples in the branches of a tree, and the function resembles a tree itself (an extended concept of *fused representations*). (d) Simulate travel and schedules on top of a real subway map image. (e) Guitar Algebra: calculate whether a given configuration of notes indicate open, Barre, or major/minor chords.



*metaphors* of embodied math and symbol grounding. An example sketch interface implementation is provided, along with the description of playtesting sessions with children in Bangladesh and USA. Additionally, examples of the Common Core math curriculum adaptation are provided (appendix A), along with examples designed by us to explore the affordances further (fig. 20).

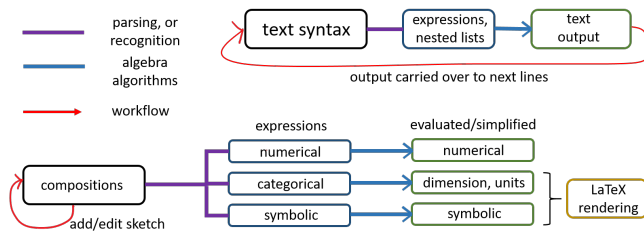
## 8.1 New Affordances for Abstract Math Activities

Traditional abstract math activities heavily rely on symbolic expressions and symbol manipulations. As demonstrated through our user studies and affordances explorations, there are ways forward to rethink abstract math in a bottom-up construction manner. We have found that the Common Core curriculum is very fitting (for most themes) for our embodied math approach.

Fused representations let us assign mathematical semantics to any drawn shapes or forms, opening up many opportunities for personalized interpretations of the same algebraic expression, which were evident in some of the freeform stories the children created. Examples such as calculating  $\pi$  (fig. 20) or guitar algebra shows that real world, physical mathematical activities can be juxtaposed with the pen and paper paradigm using our design.

## 8.2 Dependence on Digital Tools

The playtesting sessions with children also revealed that a design framework that relies on the features of interactive surfaces is not entirely intuitive for people who were never exposed to such interfaces. While this may sound like a shortcoming at first, our observations suggested that the children in those situations felt comfortable in the freeform storytelling sessions. Given the possibility of interactivity and dynamic scaffolding, we would recommend digital tools to learn mathematics.



**Figure 21: Comparison of our system (bottom) with the design of a traditional Computer Algebra System (top).**

## 8.3 Future Computer Algebra Systems

Could the future of Computer Algebra Systems (CAS) such as Mathematica or Matlab be reimagined through embodied mathematics? As fig. 21 shows, a sketch environment provides independent expression formation capabilities (as opposed to sequence-dependent expressions in CAS) anywhere in the canvas. Moreover, our design retains explanation and personalization of algebraic expressions through sketched forms. However, more research is required to make the experience fluid (e.g., using animated transitions [20]), and to design interactions for more complex expressions.

The function brush currently focuses on arithmetic operators. However, lassoing and iconic element conversion to visually define a function domain has deeper implications for advanced math. Our design framework can be adapted/extended for other mathematical primitives (such as number lines and vectors via new brush types), creating collections of them using a list brush, and using implementations of function brush that “overloads” the operators for such primitives and collections. The hierarchical nature of functions also means that custom functions can be composed for such advanced primitives. Additionally, abstraction layers would let users explore the iconic and symbolic expressions of these primitives (with an updated coercion model).

## 8.4 Compositionality

As shown in our diagrams study and embodied math metaphors, compositionality is an important characteristic for diagrams and mathematics. We have designed foldable structures that can contain visual metaphors inside one another, and the underlying coercion model allows us to form algebraic expressions from them. In this way, we have attempted to recreate a version of abstract math that the embodied math philosophy posits as “layers and layers of metaphors,” therefore our attempt is consistent with how humans think about mathematics. Going forward, more research needs to be done to evaluate this approach to math, and how domains outside of arithmetic and standard algebra can benefit from such an approach.

## 8.5 Reimagining Abstract Mathematics: What is Needed?

As mentioned in section 3, *grounding metaphors* are intuitive to humans, and *linking metaphors* are explicitly constructed notions that form abstract math concepts. One research question that both the HCI and mathematics community need to tackle together is whether advanced *linking metaphors* can be described in terms of *grounding metaphors*, and whether interactive interfaces can play a role here. Additionally, embodied mathematics fares poorly when it comes to certain concepts such as imaginary numbers, as they cannot be expressed through the *grounding metaphors* easily [22]. It remains a question whether any CAS that is designed based on an embodied math philosophy can represent such structures. Currently, the designed interactions allow us to formulate algebraic expressions that use standard arithmetic operators. For advanced mathematics such as calculus, we need to design intuitive interactions that auto-generate differential and integral equations. Finally, our approach is also relevant for other embodied interaction mediums, such as augmented reality. We aim to work on these questions and new relevant mediums in the future.

## REFERENCES

- [1] 2018. Apparatus: A hybrid graphics editor and programming environment for creating diagrams. (2018). <http://aprt.us/>
- [2] 2019. MAX/MSP Visual Programming Language. <https://cycling74.com/products/max-features>
- [3] 2019. Symbolism: A Computer Algebra Library for C#. <https://github.com/dharmatech/Symbolism>
- [4] Jo Boaler. 2015. Memorizers are the lowest achievers and other Common Core math surprises. <https://hechingerreport.org/memorizers-are-the-lowest-achievers-and-other-common-core-math-surprises/>.

- [5] Jared N Bott and Joseph J LaViola Jr. 2010. A pen-based tool for visualizing vector mathematics. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*. Eurographics Association, 103–110.
- [6] Richard Brown. 2018. *A modern introduction to dynamical systems*. Oxford University Press.
- [7] SageMath Documentation. 2020. The Coercion Model. <http://doc.sagemath.org/html/en/reference/coercion/sage/structure/coerce.html>
- [8] Vinod Goel. 1993. Sketches of thought: A study of the role of sketching in design problem-solving and its implications for the computational theory of mind. (1993).
- [9] Michael Gray. 2008. Sage: A New Mathematics Software System. 10, 6 (2008), 72–75. [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=4653208](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4653208).
- [10] Mark D Gross and Ellen Do. 1996. Ambiguous intentions: a paper-like interface for creative design. (1996).
- [11] Christian Grossauer, Florian Perteneder, Michael Haller, Jagoda Walny, John Brosz, Anthony Tang, and Sheelagh Carpendale. 2012. MathSketch: Designing a dynamic whiteboard for instruction contexts. (2012).
- [12] Md Abdul Halim. 2006. A comparative study of mathematics curriculum at primary level in Bangladesh and India (West Bengal). *Bangladesh Education Journal* (2006), 41.
- [13] Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-Sketch: Output-Directed Programming for SVG. *ArXiv abs/1907.10699* (2019).
- [14] Wolfram Research, Inc. 2019. Mathematica, Version 12.0. Champaign, IL, 2019.
- [15] Jennifer Jacobs, Joel Brandt, Radomir Mech, and Mitchel Resnick. 2018. Extending manual drawing practices with artist-centric programming tools. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 590.
- [16] Philip Nicholas Johnson-Laird. 1983. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Number 6. Harvard University Press.
- [17] Seokbin Kang, Ekta Shokeen, Virginia Byrne, Leyla Norooz, Elizabeth Bonsignore, Caro Williams-Pierce, and Jon E. Froehlich. 2020. ARMath: Augmenting Everyday Life with Math Learning. In *CHI '20*.
- [18] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: Sketching Dynamic and Interactive Illustrations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) (*UIST '14*). 395–405. <https://doi.org/10.1145/2642918.2647375>
- [19] Rubaiat Habib Kazi, Tovi Grossman, Hyunmin Cheong, Ali Hashemi, and George W Fitzmaurice. 2017. DreamSketch: Early Stage 3D Design Explorations with Sketching and Generative Design. In *UIST*, Vol. 14. 401–414.
- [20] Younghoon Kim, Michael Correll, and Jeffrey Heer. 2019. Designing Animated Transitions to Convey Aggregate Operations. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 541–551.
- [21] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks—a publishing format for reproducible computational workflows.
- [22] George Lakoff and Rafael Núñez. 2000. *Where mathematics comes from*. Vol. 6.
- [23] James A Landay. 1996. SILK: sketching interfaces like crazy. In *Conference companion on Human factors in computing systems*. ACM, 398–399.
- [24] Joseph J LaViola Jr and Robert C Zeleznik. 2004. MathPad 2: a system for the creation and exploration of mathematical sketches. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 432–440.
- [25] Bongshin Lee, Rubaiat Habib Kazi, and Greg Smith. 2013. SketchStory: Telling more engaging stories with data through freeform sketching. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2416–2425.
- [26] Bongshin Lee, Greg Smith, Nathalie Henry Riche, Amy Karlson, and Sheelagh Carpendale. 2015. SketchInsight: Natural data exploration on interactive whiteboards leveraging pen and touch interaction. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 199–206.
- [27] James Lin, Mark W Newman, Jason I Hong, and James A Landay. 2001. DENIM: an informal tool for early stage web site design. In *CHI'01 Extended Abstracts on Human Factors in Computing Systems*. ACM, 205–206.
- [28] Richard Liska, Ladislav Drska, Jiri Limpouch, Milan Sinor, Michael Wester, and Franz Winkler. 1999. *Computer algebra, algorithms, systems and applications*.
- [29] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. 2018. Data Illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 123.
- [30] Miroslaw Majewski. 2005. *Getting Started with MuPAD*. Springer-Verlag, Berlin, Heidelberg.
- [31] Maplesoft, a division of Waterloo Maple Inc.. 2019. *Maple*. Waterloo, Ontario. <https://hadoop.apache.org>
- [32] MATLAB. 2010. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts.
- [33] Maxima 2020. . System for the manipulation of symbolic and numerical expressions. Includes a collection of special functions. Utilizes exact fractions, arbitrary precision integers, and arbitrary precision floating point numbers.
- [34] Michael J. McGuffin and Christopher P. Fuhrman. 2020. Categories and Completeness of Visual Programming and Direct Manipulation. In *AVI '20*. Article 7, 8 pages. <https://doi.org/10.1145/3399715.3399821>
- [35] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Stépán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (Jan. 2017), e103. <https://doi.org/10.7717/peerj-cs.103>
- [36] Letitia Meynell. 2008. Why Feynman diagrams represent. *International Studies in the Philosophy of Science* 22, 1 (2008), 39–59.
- [37] Roy D Pea. 1987. Logo programming and problem solving. (1987).
- [38] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (Nov. 2009), 60–67. <https://doi.org/10.1145/1592761.1592779>
- [39] Hugo Romat, Nathalie Henry Riche, Ken Hinckley, Bongshin Lee, Caroline Appert, Emmanuel Pietriga, and Christopher Collins. 2019. ActiveInk: (Th)inking with Data. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) (*CHI '19*). ACM, New York, NY, USA, Article 42, 13 pages. <https://doi.org/10.1145/3290605.3300272>
- [40] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.
- [41] Jeremy Scott and Randall Davis. 2013. Physink: sketching physical behavior. In *Proceedings of the adjunct publication of the 26th annual ACM symposium on User interface software and technology*. ACM, 9–10.
- [42] The GAP Group 2019. *GAP – Groups, Algorithms, and Programming, Version 4.10.2*. The GAP Group. <https://www.gap-system.org>
- [43] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. 2018. \$Q\$: a superquick, articulation-invariant stroke-gesture recognizer for low-resource devices. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*. ACM, 23.
- [44] Bret Victor. 2011. Explorable Explanations. (2011). <http://worrydream.com/ExplorableExplanations/>
- [45] Bret Victor. 2011. Kill Math. <http://worrydream.com/KillMath/>
- [46] Bret Victor. 2011. Scrubbing Calculator. (2011). <http://worrydream.com/ScrubbingCalculator/>
- [47] Bret Victor. 2011. Up and Down the Ladder of Abstraction. <http://worrydream.com/LadderOfAbstraction/>
- [48] Bret Victor. 2012. Tangle: Explorable Explanations Made Easy. (2012). <http://worrydream.com/Tangle/>
- [49] Martin Wattenberg. 2001. Sketching a graph to query a time-series database. In *CHI'01 Extended Abstracts on Human factors in Computing Systems*. ACM, 381–382.
- [50] Heinz Werner and Bernard Kaplan. 1963. Symbol formation. (1963).
- [51] Stephen Wolfram. 2017. What is a Computational Essay. (2017). <https://blog.stephenwolfram.com/2017/11/what-is-a-computational-essay/>
- [52] Stephen Wolfram. 2020. Finally We May Have a Path to the Fundamental Theory of Physics, and It's Beautiful. <https://writings.stephenwolfram.com/2020/04/finally-we-may-have-a-path-to-the-fundamental-theory-of-physics-and-its-beautiful/>
- [53] Sally PW Wu and Martina A Rau. 2019. How students learn content in Science, Technology, Engineering, and Mathematics (STEM) through drawing activities. *Educational Psychology Review* 31, 1 (2019), 87–120.
- [54] Haijun Xia, Nathalie Henry Riche, Fanny Chevalier, Bruno De Araujo, and Daniel Wigdor. 2018. DataInk: Direct and creative data-oriented drawing. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 223.
- [55] Katherine Ye, Wode Ni, Max Krieger, Dor Ma'ayan, Jenna Wise, Jonathan Aldrich, Joshua Sunshine, and Keenan Crane. 2020. Penrose: From Mathematical Notation to Beautiful Diagrams. *ACM Trans. Graph.* 39, 4, Article 144 (July 2020), 16 pages. <https://doi.org/10.1145/3386569.3392375>
- [56] Robert Zeleznik, Andrew Bragdon, Ferdi Adeputra, and Hsu-Sheng Ko. 2010. Hands-on math: a page-based multi-touch and pen desktop for technical work and problem solving. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*. ACM, 17–26.
- [57] Bo Zhu, Michiaki Iwata, Ryo Haraguchi, Takashi Ashihara, Nobuyuki Umetani, Takeo Igarashi, and Kazuo Nakazawa. 2011. Sketch-based dynamic illustration of fluid systems. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 134.

## A COMMON CORE MATH EXAMPLES

In this appendix, we demonstrate some examples from the US Common Core mathematics curriculum, reimaged using the three brushes (iconic elements, list, function) and associated interactions presented in the design framework. The sections presented are: Counting and Cardinality (CC), Operations and Algebraic Thinking (OA), Measurement and Data (MD), Numbers in Base Ten (NBT), and Statistics and Probability (SP). We also show the approximate percentage coverage provided by our design framework.

*Methodology.* We worked through each chapter described above in the Massachusetts Common Core guideline for teachers. The guideline provides the main concepts behind each theme and corresponding example activities for students. For every example activity/problem, we worked to faithfully recreate the concept using the tools afforded by our design framework. In cases when an example explicitly asks for different representations that are absent in our design (such as number line, line graphs), or use geometric primitives that are currently unsupported in our design, we marked it as "not applicable N/A". A few of these examples were playtested with children.

To calculate the coverage, we simply look at the percentage of examples covered compared to the total number of examples under a theme.

### A.1 Coverage

Exercises under the SP theme often had representations that are unavailable in our system, therefore the SP coverage is lower than other themes. However, for many other themes, our design framework captures the essence of the proposed exercises in most cases (fig. 22).

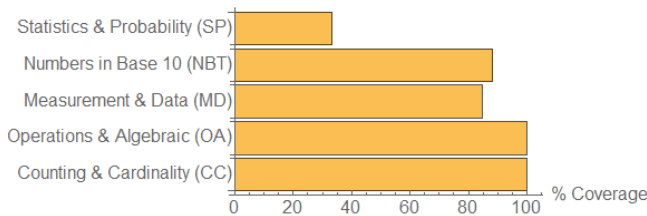


Figure 22: Percentage coverage of different sections from the Common Core curriculum.

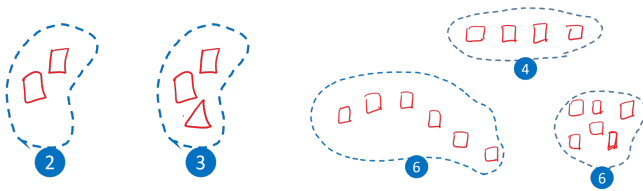


Figure 23: Some Counting and Cardinality (CC) examples.

### A.2 Counting and Cardinality

In Kindergarten and first grade, counting and cardinality exercises typically ask to categorize and arrange objects in different spatial configurations, and compare the cardinalities (fig. 23).

### A.3 Operations and Algebraic Thinking (OA)

Common exercises include understanding the associativity, distributivity, commutativity of arithmetic operations (fig. 24). Using hierarchical lists and functions compositions, and by exchanging variables in and out of intermediate structures, the invariance of such rules can be demonstrated. Understanding arithmetic patterns are also encouraged from the second grade onward. Here, we show two adapted examples from third and fourth grades, where moving lists or iconic elements in or out would produce invariant patterns. The guideline explicitly asks for "visual or drawn models" to demonstrate these concepts.

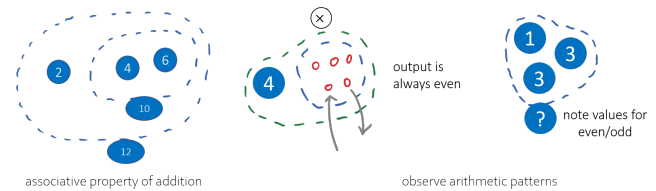


Figure 24: Some Operations and Algebraic Thinking (OA) examples.

### A.4 Numbers in Base Ten (NBT)

Starting from first grade, moving on until fifth grade, NBT exercises often ask students to construct bigger numbers using unit "bundles" of numbers. Our design is apt for such hierarchical structures. Using hierarchies of lists, one could define such "bundles", name them, and use them to compose bigger numbers (fig. 25).

### A.5 Statistics and Probability (SP)

Many SP exercises explicitly use the number line representation, which is currently unavailable in our design. However, there are quite a few exercises that ask students to construct simple functions such as mean, sum of sums etc. One such function construction is shown in fig. 26 where the exercise asks students to calculate mean height of some drawn figures. Using the "number of elements" property of a function, one can easily construct simple functions such as average.

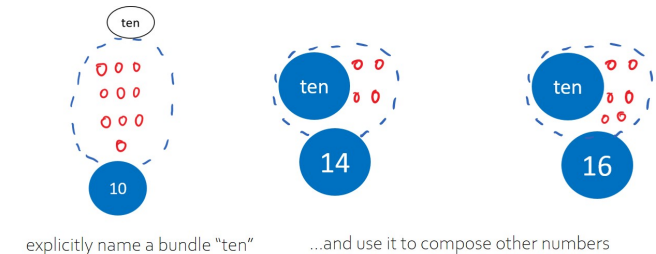


Figure 25: Some Numbers in Base Ten (NBT) examples.

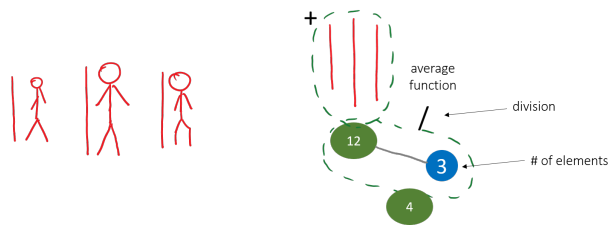


Figure 26: A Statistics and Probability (SP) example.

### A.6 Measurement and Data (MD)

Many MD examples are well suited with our design. Examples in fig. 27 are: (a) In the first grade, many of the exercises ask students to measure something using unit lengths of another straight object. In Noyon, real world objects (e.g., a matchstick) can be put on top of the screen and measured by drawing a line, and later, the line can be copied and arranged to measure bigger objects (e.g., a book's dimensions). (b) The "container" metaphor (i.e., hierarchical structures) are particularly suitable for exercises involving measuring units, money etc. This is an example taken from the third grade syllabus, where students are asked to work with smaller units of money and compose bigger units from them. A student can draw the cents, dimes etc. in any shape and form they want.

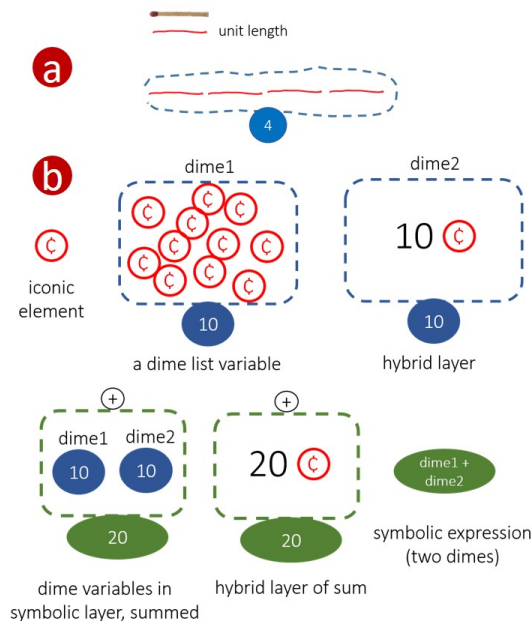


Figure 27: Some Measurement and Data (MD) examples.

### B SOME EXAMPLES OF FREEFORM STORIES MADE BY CHILDREN

See Figure 28. These stories were created during the freeform playtesting sessions in both Bangladesh and USA.

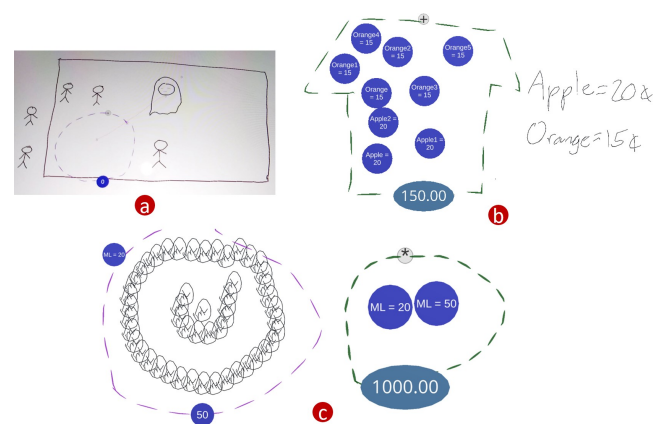


Figure 28: (a) Ghost story (employing nested lists and graphical transforms), (b) composing custom units of money ("apple" and "orange") to find out how much money was spent in a grocery shop. (c) Creating iconic elements in the shape of droplets, one child called each droplet a "ml" and created a list of 50 "ml" of water in this way. Later, he created one more list, and created 1 "liter" (1000 "ml") of water using multiplication.