SUBSTANCE and STYLE: domain-specific languages for mathematical diagrams

Wode Ni *
Columbia University

Katherine Ye* Joshua Sunshine
Jonathan Aldrich Keenan Crane
Carnegie Mellon University

Abstract

Creating mathematical diagrams is essential for both developing one's intuition and conveying it to others. However, formalizing diagrams in most general-purpose tools requires painstaking low-level manipulation of shapes and positions. We report on early work on PENROSE, a system we are building to automatically visualize mathematics from notation. PENROSE comprises two languages: SUBSTANCE, a domain-specific language that mimics the declarativeness of mathematical notation, and STYLE, a styling language that concisely specifies the *visual semantics* of the notation. Our system can automatically visualize set theory expressions with user-defined styles, and it can visualize abstract definitions of functions by producing concrete examples. We plan to extend the system to more domains of math. [1]

1. Separating SUBSTANCE and STYLE

To formalize a mathematical diagram in a general-purpose tool, one must lower one's high-level domain-specific understanding of the diagram's semantics into the weeds of styling details. A general-purpose illustration tool like TikZ has no notion of the domain a user is trying to illustrate, so it exposes total control over display attributes such as shape, color, and position. At this level, it is very difficult to illustrate multiple objects in a domain without duplicating code, and difficult to cleanly illustrate an object in different ways.

To raise the level of abstraction for creating illustrations, users often define parametrized macros or write libraries to translate domain-level descriptions to diagrams. For example, TikZ users have created thousands of ad-hoc domain-specific languages to model domains like commutative diagrams and Bayesian networks. However, these DSLs lack the affordances of full-blown programming languages, such as domain-level type-checking and consistent syntax.

PENROSE aims to provide a principled, extensible domain-specific illustration environment for mathematics. To enable users to create and edit diagrams at a semantic level, PENROSE enforces a clean separation between substance and style. This separation takes the form of two languages, SUBSTANCE and STYLE, akin to HTML and CSS. SUBSTANCE models mathematical notation, and STYLE defines the *visual semantics* of the notation: how the objects and relationships declared in SUBSTANCE are translated into images.

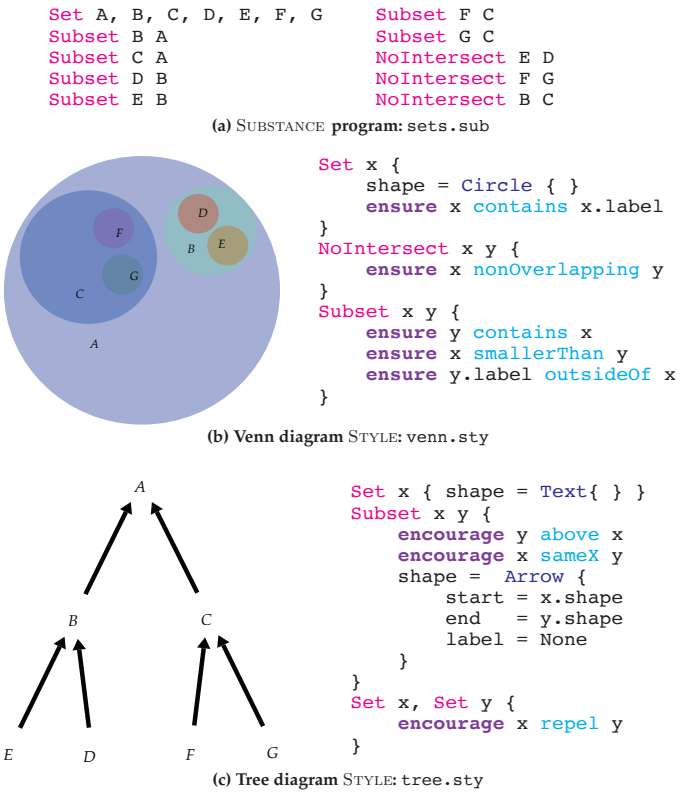

Figure 1. Two STYLEs visualizing the same SUBSTANCE.

Figure 1 shows how one might use PENROSE in the domain of set theory. Here, we illustrate one set theory expression with two visual representations. The SUBSTANCE program at the top of Figure 1 specifies the set objects and their relationships. The two STYLE programs emphasize different aspects of the relationship $A \subset B$, which is commonly interpreted either as "A is spatially contained in B" or "A implies B" (because if a point lies in A, then it lies in B). The styles are generic, and thus reusable: they apply to any SUBSTANCE program that includes sets and relationships.

* These authors contributed equally to this work.

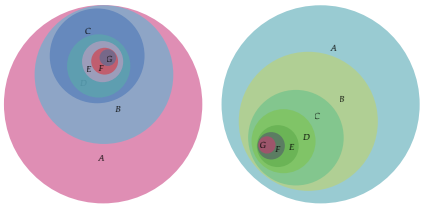

Figure 2. Automatic layout of nested sets.

2. Declaring the display with STYLE

Many problems in visualization can be cast as optimization problems. Instead of hand-designing an algorithm to place labels just right, one can simply write a function that takes a diagram state and returns a number that measures the badness of the label placement. This badness function, typically called an *objective*, can then be minimized using optimization methods such as gradient descent, yielding a close-to-optimal label placement or diagram state.

The design of STYLE leverages the insight that optimization problems can be stated declaratively and solved generally. In a STYLE program, the user applies individual smaller objectives to selected parts of a diagram, which the PENROSE compiler composes into an overall objective function measuring the badness of the entire diagram. This objective is minimized by the PENROSE runtime, which can lay out a tree, a circle-packing, or indeed any layout that can be expressed as an objective. Using STYLE enables the user to tap into the power of optimization-based layout without being an expert in optimization or layout algorithms.

Consider the nested sets depicted in Figure 2. To create those diagrams, a TikZ user would have to manually specify the sets' sizes and positions and ensure that every subset is smaller than its container set. A PENROSE user can simply create these diagrams by specifying which sets are subsets of other sets in a SUBSTANCE program, then write a STYLE program that reads like plain English.

```
Subset x y {
  ensure x smallerThan y
}
```

The system compiles the style into an overall objective function and can generate several layouts, shown in Fig. 2.

In general, the user can leverage STYLE's pattern-matching and binding mechanisms to select objects, which encourages the resulting styles to be generic and reusable. Once an object or relationship is selected, the user can concisely specify its visual instantiation, as well as visual relationships between selected objects.

We envision that libraries of STYLE programs will encode the design expertise of skilled illustrators for all to read, use, and remix. Because PENROSE can optimize any attribute that can be expressed in terms of objectives, a creative user can write styles that fine-tune diagram attributes beyond position and size, such as global angle alignment and color harmony.

3. Illustrating abstract function definitions

Understanding functions is crucial to understanding mathematics. Many of the first definitions a learner encounters in elementary discrete math involve properties of functions, and these definitions continue to show up in more advanced domains like topology and category theory.

One common object of study is the injective function. A function is injective, or "one-to-one," if every element of its codomain is mapped to at most one element of the domain. That is, a function $f : X \to Y$ is injective if:

$$\forall x, x' \in X, f(x) = f(x') \to x = x'.$$

People often illustrate abstract definitions by example. In this case, one common "cartoon," which appears in textbooks and on Wikipedia, stylizes f as a set of mappings between elements of discrete and finite sets.

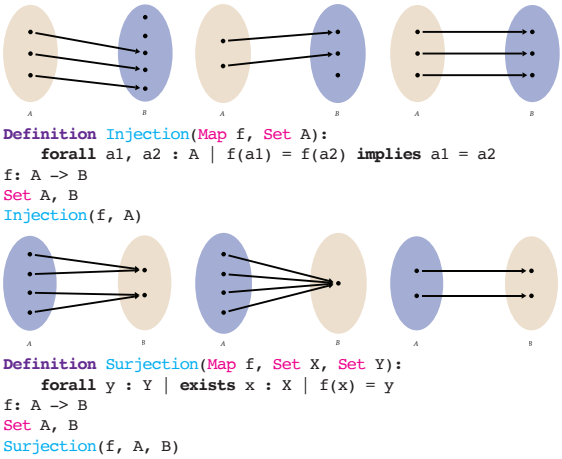

Figure 3. Visualizing injective and surjective functions.

Given only this abstract definition of an injective function, written in SUBSTANCE, PENROSE can automatically visualize injective functions in the same style (Figure 3). In addition, PENROSE can automatically generate many concrete examples of injective functions, giving the reader a better intuition for the definition (*e.g.* bijections are injections).

PENROSE can automatically visualize any function definition on discrete, finite sets that is written in first-order logic, such as function composition. The runtime finds concrete instances by calling the external tool Alloy [2].

We plan to design principled mechanisms for users to extend PENROSE with domain knowledge. Users should be able to incorporate external tools, like Alloy, and external libraries, such as existing graph layout libraries, in every stage of creating a visualization. For more information on PENROSE, visit `http://www.penrose.ink`.

References

[1] Ye, Katherine, et al. *Designing extensible, domain-specific languages for mathematical diagrams.* Off the Beaten Track, 2017.

[2] Jackson, Daniel. *Software abstractions.* MIT Press, 2012.