Designing extensible, domain-specific languages for mathematical diagrams

Katherine Ye Keenan Crane Jonathan Aldrich Joshua Sunshine
Carnegie Mellon University
{kqy, kmcrane, jonathan.aldrich, sunshine}@cs.cmu.edu

1. Motivation for the PENROSE system

In science, a well-chosen illustration can turn bafflement into enlightenment. Yet technical exposition remains largely textual, due to the tremendous expertise required to produce high-quality figures. For example, of the fifty newest mathematical papers submitted to arXiv at the time of this writing, only one-third of them had figures.

To illustrate books and talks, professionals often turn to a diagram language that integrates with TEX. However, languages such as TikZ and Asymptote require manipulating graphical primitives at a low level, often by specifying coordinates, styles, and labels of objects. The alternative, using GUI-based software, requires a painfully manual workflow. To draw a simple torus mesh with shading and a normal vector, illustrators must build a 3D model, hand-trace it in vector graphics software, and hand-shade and label it. They must redo that process for even substantially similar diagrams.

For the domain of mathematical diagrams, a high-level, declarative language already exists to describe what users desire to illustrate: the language of mathematical notation. Users ought to be able to create and manipulate their diagrams at this level. Thus, we propose to create PENROSE,[1] a system to automatically generate professional-quality mathematical illustrations from high-level, purely semantic descriptions of mathematical objects. (Our team possesses domain expertise, as one of the authors is an expert in illustrating mathematics.) Unlike low-level tools such as Adobe Illustrator, where diagrams are specified via graphical primitives, a mathematically-inclined user *should not require **any** graphic design skill to create beautiful diagrams*.

PENROSE comprises two extensible domain-specific languages (DSLs): SUBSTANCE, which diagrammers use to specify mathematical objects and relationships, and STYLE, which implementers use to encode various ways of realizing these relationships visually, akin to the separation of content and style in modern HTML/CSS. To compile diagrams, we are developing a sophisticated constraint solver incorporating techniques from optimization and computer graphics.

[1] ...after Sir Roger Penrose, who, in addition to possessing a euphonious name, is known for his Escher-inspired illustrations of impossible objects.

Set A, B, C
Point p
$p \in A \cap B \cap C$

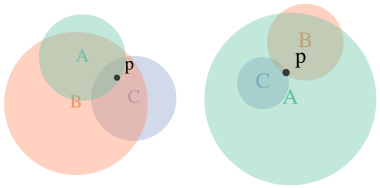

Figure 1. A SUBSTANCE program specifying that a point lies in the intersection of three sets, and two algorithmically-generated PENROSE visualizations of this program.

2. Three examples

Set theory. In Figure 1, we give a SUBSTANCE program in a "sub-DSL" for set theory. This program mimics the declarative notation that would appear in a textbook. How should PENROSE illustrate it? One approach is to view the program as a system of constraints. The system samples primitives until they satisfy these constraints, then picks good final diagrams by minimizing an energy function. Intelligently sampling diagrams unearths interesting corner cases of a specification, akin to fuzzing mathematics. Users would likely not think first of cases where one set is contained in another.

Extending set theory with group theory. A group is often introduced as a set closed under a binary, associative operation, where the set contains an identity element with an inverse. More intuitively, a group can also be seen as a collection of actions, where every action is reversible and every sequence of actions is also an action [1]. Twisting a Rubik's cube forms a group; so does just rotating a rectangle.

It is natural for mathematical exposition to mention groups and sets in the same breath. Here is a description of a particular group from Wikipedia: "The Klein four-group is defined by the group presentation $\mathrm{V} = \langle a, b \mid a^2 = b^2 = (ab)^2 = 1 \rangle$... Another numerical construction of the Klein four-group is the set $\{1, 3, 5, 7\}$, with the operation being multiplication modulo 8. Here a is 3, b is 5, and $c = ab$ is $3 \cdot 5 = 15 \equiv 7 \mod 8$." Thus, to design SUBSTANCE, we take note: mathematicians often introduce similar objects at different levels of abstraction. A group may be described with a given name (the Klein 4-group, or K_4), a popular no-

```
Group k4 = FromPresentation(<r, b | r^2 = b^2 = (rb)^2 = e>)
k4.set({e = 1, r = 3, b = 5, rb = 7})
k4.op(\x y -> x * y `mod` 8)

AbstractDiagram k4set = Venn(k4.set, name = "K4")
AbstractDiagram k4cayley = Cayley(k4)
CompositeDiagram linked = link(k4set, k4cayley,
     name = "multiplication mod 8")

linked.layout = left_to_right -- not specifying coordinates!
k4set.e.style = Circle {r = 10, color = green} -- the identity
linked.e.style = dotted
```

Figure 2. Linking two views of the Klein 4-group.

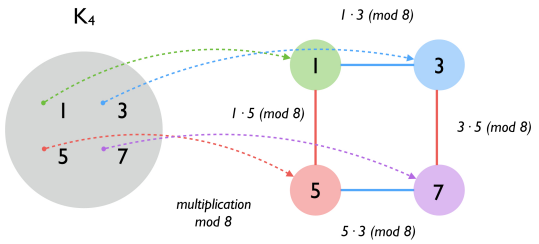

Figure 3. A mockup of a possible PENROSE visualization.

tation (its group presentation[2] $\langle a, b | a^2 = b^2 = (ab)^2 = 1 \rangle$), or a set and an operation on it ($(\{1, 3, 5, 7\}, \cdot \mod 8)$).

To make the Wikipedia explanation more concrete, we visualize the connection between the Klein four-group as a set with an operation and as exploratory actions. We give a combined SUBSTANCE and STYLE program in Figure 2. The first three lines instantiate the abstract group $k4$ with a concrete set and operation. The next three lines introduce two different *views* of the group: as a Venn diagram, from the Set sub-DSL, and as a Cayley diagram[3] depicting actions in the group, from the Group sub-DSL. A *view* bridges content and style by specifying the abstract structure of how an object is displayed, but not its on-screen attributes.

Geometry. We will extend PENROSE to handle issues of connectivity and geometry arising in domains such as graph theory and topology, as in Figure 4. Rendering 3D geometry as high-quality vector graphics raises new challenges at the intersection of computer-aided design and 2D illustration.

3. System and language design questions

Modeling mathematics. Users want to diagram their own exotic domains. Thus, STYLE and SUBSTANCE should be extensible with sub-DSLs with user-defined semantics and graphics. Users will want to illustrate relationships between objects in different domains, as in theorems and proofs, or maps from one domain to another. How can we enable these sub-DSLs to be productively used together? Additionally,

[2] It gives the group *generators* (here, a and b) and relates the generators to each other and the identity so that the group is "uniquely determined."

[3] Cayley diagrams are widely used to create almost-cartographic maps of groups. Starting from the identity, one combines a generator with an element of the group, using the group operation, to reach another element. The action is shown as an edge, and the result is shown as a state.

```
S := Sphere(2)
T := Torus(2)
I := [0,1] subset Reals
f := T -> S
gamma := I -> S
gamma(0) = gamma(1)
curve1 := Image(gamma)
curve2 := PreImage(curve1, f)
Homotopic(curve2, 0) = FALSE
p := Point in curve2
```

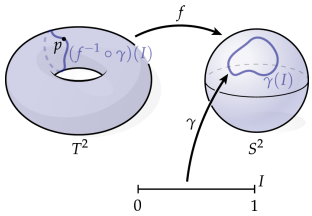

Figure 4. A map from a torus to a sphere in SUBSTANCE.

most domains of mathematics are highly connected, often hierarchically. Lie groups are both groups and manifolds: should we model them with subtyping or typeclasses?

DSL implementers vs. end-users. PENROSE users might want to build their own sub-DSL, or just program in an existing one. Thus, we divide the system into two phases, SETUP and DIAGRAM. In SETUP, an implementer may build a sub-DSL from scratch by providing its grammar, syntax, and libraries. PENROSE provides an environment in a high-level host language in which the implementer can embed their sub-DSL, and it will check the sub-DSL at compile-time. In DIAGRAM, an end-user writes programs in this sub-DSL, which are again typechecked at compile-time. Existing work in quasiquoting [3] addresses the problem of type-checking an embedded DSL *and* a program in that DSL at compile-time. We could, say, embed PENROSE in Template Haskell and implement our compilers as Haskell code generators.

Designing STYLE. How should we represent styles at different levels of abstraction? In the group theory example discussed in Section 2, users may want to import styles from some standard library. Some styles are specific to K_4; other styles are generic over all groups; yet others are generic over all domains of math. Also, should diagram styles be specified declaratively, as in CSS, or algorithmically, since an object may demand a specialized drawing algorithm?

Interaction. Using SUBSTANCE seems to require giving up granular control over layout. In TikZ, users cannot edit diagrams by directly manipulating the diagram, as is natural–they must reverse-engineer the desired changes in the program. However, we plan to build on the *"prodirect"* paradigm [2] to combine the strengths of programmatic and direct manipulation. Users might make low-level tweaks not at the program level, but by dragging a node in a GUI. How should PENROSE infer the corresponding program changes? Also, generating diagrams programmatically opens the door to generating interactive diagrams, such as parametrized ones. To visualize a cycle graph of n vertices, the user could "scrub" the parameter n over concrete values.

References

[1] N. Carter. *Visual group theory.* MAA, 2009.

[2] R. Chugh, B. Hempel, M. Spradlin, and J. Albers. Programmatic and direct manipulation, together at last. In *PLDI '16*.

[3] G. Mainland. Quasiquoting for Haskell. In *Haskell '07*.