Substance and Style: domain-specific languages for mathematical diagrams

Wode “Nimo” Ni
Columbia University
with Katherine Ye, Joshua Sunshine, Jonathan Aldrich, Keenan Crane
Diagram from Wikipedia page for “sine”. https://commons.wikimedia.org/wiki/File:Circle_cos_sin.gif
Illustrating Venn diagram in TikZ

\documentclass{article}
\usepackage{tikz}
\begin{document}
\pagestyle{empty}
\begin{tikzpicture}
    \begin{scope}[shift={(3cm,-5cm)}, fill opacity=0.5]
        \draw[fill=red, draw = black] (0,0) circle (5);
        \draw[fill=green, draw = black] (-1.5,0) circle (3);
        \draw[fill=blue, draw = black] (1.5,0) circle (3);
        \node at (0,4) (A) {\large\textbf{A}};
        \node at (-2,1) (B) {\large\textbf{B}};
        \node at (2,1) (C) {\large\textbf{C}};
        \node at (0,0) (D) {\large\textbf{D}};
    \end{scope}
\end{tikzpicture}
\end{document}
Illustrating Venn diagram in TikZ

\documentclass{article}
\usepackage{tikz}
\begin{document}
\pagestyle{empty}
\begin{tikzpicture}
  1. Specify the colors, sizes, and locations of the circles
  2. Create “nodes” for the labels and determine the locations of the labels
\end{tikzpicture}
\end{document}
Illustrating Venn diagram in TikZ

\documentclass{article}
\usepackage{tikz}
\begin{document}
\pagestyle{empty}
\begin{tikzpicture}
\begin{scope}[shift={(3cm,-5cm)}, fill opacity=0.5]
\draw[fill=red, draw = black] (0,0) circle (5);
\draw[fill=green, draw = black] (-1.5,0) circle (3);
\draw[fill=blue, draw = black] (1.5,0) circle (3);
\node at (0,4) (A) {\large\textbf{A}};
\node at (-2,1) (B) {\large\textbf{B}};
\node at (2,1) (C) {\large\textbf{C}};
\node at (0,0) (D) {\large\textbf{D}};
\end{scope}
\end{tikzpicture}
\end{document}
Illustrating Venn diagram in TikZ

Low-level manipulation down to pixels.

Semantics of the diagram is completely lost.
What were we trying to illustrate originally? Circles and text labels?
Graph visualization using Graphviz

```plaintext
graph G {
  e
  subgraph clusterA {
    a -- b;
    subgraph clusterC {
      C -- D;
    }
  }
  subgraph clusterB {
    d -- f
  }
  d -- D
  e -- clusterB
  clusterC -- clusterB
}
```
Graph visualization using Graphviz

High-level and clean, but only works for graphs!
Direct manipulation tools

If
Pr[X \ B] = Pr[Y \ B]
then
| Pr[X] − Pr[Y] | ≤ Pr[B]
Language design
**Penrose** has two extensible DSLs

<table>
<thead>
<tr>
<th>Substance</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-level declaration of mathematical objects</td>
<td>Specifies the visual <em>presentation</em> of the objects</td>
</tr>
</tbody>
</table>

```plaintext
Set A, B
Intersect A B

Set A {
    shape = Circle{}
}

Intersect X Y {
    ensure X overlapping Y
}
```
Set A, B
Intersect A B

Set A {
    shape = Circle{}
}
Intersect X Y {
    ensure X overlapping Y
}
Substance and Style

- Power users encode domain expertise in Style
- Casual users create diagrams declaratively using Substance
Dr. M wants to visualize a function…

How should I draw $f : X \rightarrow Y$?
Dr. M wants to visualize a function…

How should I draw $f : X \to Y$?
Dr. M wants to visualize a function…

How should I draw $f : X \rightarrow Y$?
Dr. M wants to visualize a function...

How should I draw $f: X \rightarrow Y$?
Dr. M wants to visualize a function…

Abstract .sty

Cartesian .sty

\[ f(x) \]
Style language

Set $X$ {
  shape = Circle {
    color = blue
  }
}

Selectors using pattern matching
Style language

Set X {
  shape = Circle {
    color = blue
  }
}

Subset X Y {
  X.color = green
}

Cascading: later rules override the earlier ones
Style language

Set X {
    shape = Circle {
        color = blue
    }
}

Subset X Y {
    X.color = green
}

Set `A` {
    A.color = pink
}
Style language

`Set X { 
    shape = Circle { 
        color = blue 
    } 
} 
Subset X Y { 
    X.color = green 
} 
Set `A` { 
    A.color = pink 
}

Wait. How does Penrose choose the positions and sizes of them?
Style language

Set X {
    shape = Circle {
        color = blue
    }
}

Subset X Y {
    X.color = green
}

Set `A` {
    A.color = pink
}

*Wait.* How does Penrose choose the *positions* and *sizes* of them?
Optimization-based layout
Specifying Layout in Style

Set X {
  shape = Circle {
    color = blue
  }
}

Subset X Y {
  X.color = green
  ensure Y contains X
  ensure X smallerThan Y
  ensure Y.label outsideOf X
}

Declaratively specify optimization requirements!
Optimizing diagram layout

![Diagram showing the relationship between Energy and Time. The graph illustrates a decreasing trend as time progresses.]
Optimizing diagram layout

Exterior point method + line search

Energy

Time
Illustrating abstract function definitions
A function is **injective**, or “**one-to-one,**” if every element of its codomain is mapped from *at most* one element of the domain.

\[ \forall x, x' \in X, f(x) = f(x') \rightarrow x = x'. \]

**Definition** Injection(Map f, Set A):

forall a1, a2 : A | f(a1) = f(a2) implies a1 = a2
Extend with external tools

• Penrose doesn’t have to understand first order logic!

• Extend the system to integrate with an external tool, Alloy, to generate concrete instances given the definition of injective functions.

**Definition** Injection(Map f, Set A):
  
  \[
  \forall a_1, a_2 : A \mid f(a_1) = f(a_2) \implies a_1 = a_2
  \]

f: A → B
Set A, B
Injection(f, A)

Penrose program
Extend with external tools

- Penrose doesn’t have to \textit{understand} first order logic!
- Extend the system to integrate with an \textbf{external tool}, \textit{Alloy}, to generate concrete instances given the definition of injective functions.

\textbf{Definition} \textbf{Injection}\(\text{(Map } f, \text{ Set } A)\): \\
\begin{align*}
\text{forall } a1, a2 : A & | f(a1) = f(a2) \implies a1 = a2 \\
f & : A \rightarrow B \\
\text{Set } A, B \\
\text{Injection} (f, A)
\end{align*}

Penrose program
Extend with external tools

• Penrose doesn’t have to understand first order logic!
• Extend the system to integrate with an external tool, Alloy, to generate concrete instances given the definition of injective functions.
Extend with external tools

• Penrose doesn’t have to understand first order logic!

• Extend the system to integrate with an external tool, Alloy, to generate concrete instances given the definition of injective functions.

```alloy
sig A {
  f : B
}
sig B {
}
fact {
  all a1,a2 : A | a1.f = a2.f implies a1 = a2
}
pred show() { } 
run show for 5
```
Extend with external tools

• Penrose doesn’t have to understand first order logic!

• Extend the system to integrate with an external tool, Alloy, to generate concrete instances given the definition of injective functions.
Extend with external tools

• Penrose doesn’t have to *understand* first order logic!

• Extend the system to integrate with an **external tool**, *Alloy*, to generate concrete instances given the definition of injective functions.

```
30

sig A {
    f : B
}

sig B {
}

fact {
    all a1, a2 : A | a1.f = a2.f implies a1 = a2
}

pred show() { }

run show for 5

```

All $A$: \{A$0$\}

$B$: \{B$0$, B$1$, B$2$, B$3$, B$4$\}

$f$: \{A$0$→B$4$\}

Alloy instances
Extend with external tools

• Penrose doesn’t have to understand first order logic!

• Extend the system to integrate with an external tool, Alloy, to generate concrete instances given the definition of injective functions.
Extend with external tools

- Penrose doesn’t understand first order logic!
- Extend the system to integrate with an external tool, Alloy, to generate concrete instances given the definition of injective functions.
Visualizing injective functions
Visualizing injective functions
Visualizing injective functions
A **bijection** is a special case of injection!
Visualizing composition of functions

Set $A, B, C$

$f: A \rightarrow B$

$g: B \rightarrow C$

$\text{Injection}(f, A)$

$\text{Injection}(g, B)$
Visualizing composition of functions

Set $A, B, C$

$f: A \rightarrow B$

$g: B \rightarrow C$

Injection$(f, A)$

Injection$(g, B)$

The composition of injections is still an injection!
Running Penrose
Example: Venn vs Tree diagram

Set A, B, C, D, E, F, G
Subset B A
Subset C A
Subset D B
Subset E B
Subset F C
Subset G C
NoIntersect E D
NoIntersect F G
NoIntersect B C
Example: Venn vs Tree diagram
Example: Venn vs Tree diagram
Example: restarting the optimization
Conclusion

- **Goal**: Enables users to make beautiful diagrams **without** design background
- **DSLs** that cleanly separate high-level semantics and styling details
- Optimizes layout declaratively
- Extensible system

[http://penrose.ink](http://penrose.ink)
[https://github.com/penrose/penrose](https://github.com/penrose/penrose)