Language Specifications

Sammy Sasaki, Mico Mendoza

0.1 Introduction

Geometric abstraction is an abstract art form that primarily uses simple geometric shapes painted with flat colors and placed in non-illusionistic planar space. Many people enjoy geometric abstract art, but sometimes they can be hard to create or find, especially for people with little to no artistic skills.
This is a domain specific programming language that offers users a way to easily create geometric abstract art. It lets users create art that is customized to their own liking, in terms of color scheme, size, shapes, and other parameters.

0.2 Design Principles

The main guiding principles in this programming language are simplicity and usability. We want our language to be simple and easy to use, and give the user as much power as possible to create the art that they are envisioning for themselves. We designed our language so that the users have the power to easily customize the primary visual features of a geometric abstract art – the 2D geometirc shapes and colors/color scheme. Other visual features that we are keeping in mind as we develop our language further are the number of edges for each shape, the relative sizes of the shape, the positions of the shape, the colors of adjacent shapes, the size of the canvas that we are filling up with shapes (and gaps between shapes), etc.

0.3 Examples

```
5 shapes, scheme: greyscale, max 8 edges

7 shapes, scheme: rainbow, max 6 edges

2 shapes, scheme: red green blue purple, max 3 edges
```

0.4 Language Concepts

Each program takes 3 parameters to specify the geometric abstract art a programmer wants to create. The number of shapes and the number of maximum edges for any given shape are primitive types (int) in this language, while the color scheme is a combined form of the primitive type color (string), where a programmer can choose from preset color schemes or customize their own (must be 4 or more distinct colors).

0.5 Minimal Formal Syntax

```
<expr>   ::= <num> shapes, scheme: <scheme>, max <num> edges
<num>    ::= (is any positive integer)
<scheme> ::= <preset>
           | <color>4+
<color>  ::= red | orange | yellow | green | blue | indigo | purple
<preset> ::= greyscale | rainbow
```

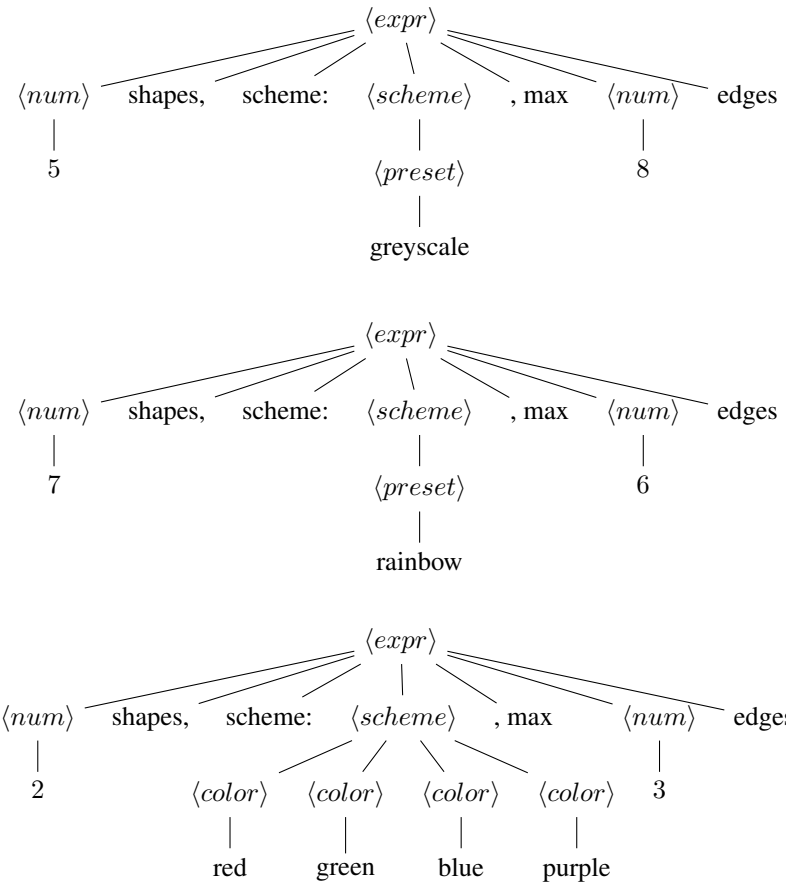
0.6 Minimal Semantics

(i) There are 2 primitive types: shapes and max edges, which are both ints.

(ii) Our program combines shapes into a connected image with no gaps.

(iii) The types will be ints for num shapes and max edges, and algebraic data type of strings for the schemes, which is another ADT that can be a preset or user inputted. The user inputted colors are also an ADT of strings for the colors.

(iv)



(v) The programs don't take any user input outside of the original program lines. They output svg files that when opened display a colored canvas containing the art piece. The evaluation would be done using svg. First a string of svg commands would be created when evaluating the tree, and that would be fed into a file which can be opened.

Rough sketch of evaluation steps: Take all the parameters. Then loop through the number of shapes. To start, choose a random color and a random number of edges from 3 to max edges. Then draw that shape in the top left corner. Then choose a random edge that is not on the edge of the canvas, a random color that was not just used, and another random number of edges, and draw another shape adjacent to the chosen edge from the first shape. Continue until all the shapes are drawn, making sure to remain inside the canvas. Then fill in the remaining parts of the canvas with random colors, ensuring that no adjacent shapes are the same color.

0.7 Limitations/Remaining Work