GeoDraw: Language Specification

Isabel Grondin and Meredith Wolf

Spring 2022

1 Introduction

GeoDraw is a language that lets you use basic math equations (lines, circles, parabolas, etc) to construct basic cartoon drawings. Shading can be done by using inequalities. This is inspired by a project in high school that was designed to help us learn more about geometry. It was difficult to find a graphing calculator online that made this particular project easy. GeoDraw will be a be a fun educational tool to strengthen students knowledge of geometrical equations.

This language can also be viewed as a gentle introduction to computer graphics, which use much more complex mathematical equations, and to programming more generally. For that reason simplicity is an important part of our language design.

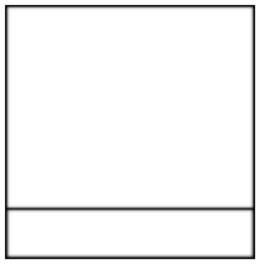
2 Design Principles

Our guiding principles are simplicity and readability. Since this language is targeted towards younger populations, who are most likely being introduced to programming, we want the language to be fairly intuitive. Furthermore, GeoDraw is a tool aiding students in learning graphing skills, and we do not want the complexity of the language to detract from that core objective.

3 Example Programs

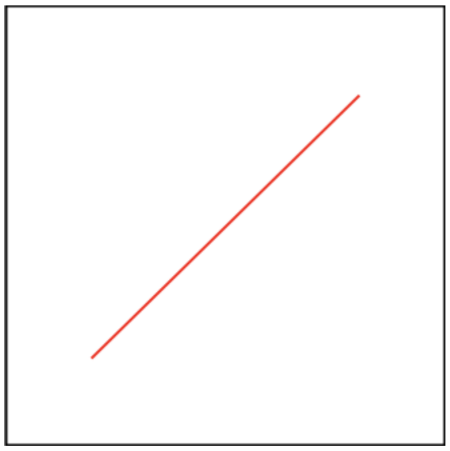
Example 1:

```
canvas(5,5)
draw(y = 1, col='#000000', brushType ='simple');
```



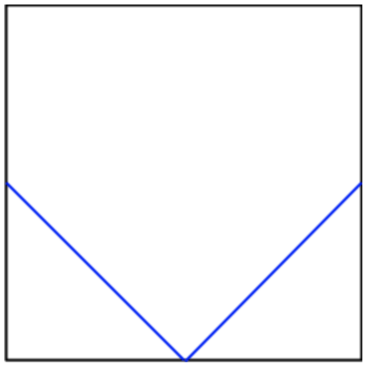
Example 2:

```
canvas(10, 10)
draw(y = x, bounds=(2<x<8), col='#ff0000', brushType ='simple')
```



Example 3:

```
canvas(8, 8)
draw(y = abs(x-4), col='#0000ff', brushType='simple')
```



4 Language Concepts

The core concept a user needs to understand for GeoDraw, is how to create an equation of the form $< Y >< eq >< Exp >$ to generate the desired line and shading effect. An Expression,$< Exp >$ consists of the primitive data types X, real numbers and operations. Draw is a combining form in our language that is made up of an equation, color, bounds, and brush style.

5 Syntax

Here is the equivalent, formal definition of the grammar in Backus-Naur form of the parts of our language we have implemented:

$<\texttt{Sequence}>$	::= $\texttt{<Canvas> <Expr>}^+$
$< Expr >$	$::=< Draw > \mid < Sequence >$
$< Draw >$	::= draw$(< ws >< Equation >< ws >, < ws >< Bound >< ws >, < ws >< Color >< ws >, < ws ><$
$< Canvas >$	$::= (< ws >< CanvasNum >< ws >, < ws >< CanvasNum >< ws >, < ws >< Color >< ws >)$
$< Equation >$	$::=< Y >< ws >< Equality >< ws >< Oper >$
$< Y >$	$::= Y \mid x$
$< X >$	$::= X \mid x$
$< Equality >$	$::=< \mid = \mid >$
$< Oper >$	$::=< Add > \mid < Sub > \mid < Div > \mid < Mult > \mid < X > \mid < Num >$
$< Sub >$	$::= (< Oper >< ws > - < ws >< Oper >)$
$< Div >$	$::= (< Oper >< ws > / < ws >< Oper >)$
$< Add >$	$::= (< Oper >< ws > + < ws >< Oper >)$
$< Mult >$	$::= (< Oper >< ws > * < ws >< Oper >)$
$< Pow >$	$::= (< Oper >< ws >^{<} ws >< Oper >)$
$< Sin >$	$::= sin(< ws >< Oper >< ws >)$
$< Cos >$	$::= cos(< ws >< Oper >< ws >)$
$< Sqrt >$	$::= sqrt(< ws >< Oper >< ws >)$
$< Abs >$	$::= abs(< ws >< Oper >< ws >)$
$< Bound >$	$::= [< SingleBound >] \mid [< BoundList >] \mid [NoBounds]$
$< SingleBound >$	$::=< ws >< Var >< ws >< Equality >< ws >< Num >< ws >$
$< BoundList >$	$::=< SingleBound >< SingleBound >^+$
$< Var >$	$::=< Xvar > \mid < Yvar > \mid VarError$
$< Xvar >$	$::= X \mid x$
$< Yvar >$	$::= Y \mid y$
$< Color >$	$::= (< ws >< Num >< ws >, < ws >< Num >< ws >, < ws >< Num >< ws >)$
$< Brush >$	$::= Simple \mid Funky \mid Thick \mid Other$
$< Num >$	$::= \mathbf{R}$
$< ColNum >$	$::= 0 \leq Int \leq 255$
$< CanvasNum >$	$::= 0 \leq Int \leq 600$
$< ws >$	$::= \textvisiblespace \mid \varepsilon$

Primitives:

- x,y
- real numbers
- colors (hex codes)
- operations(+ - / * sin cos pow sqrt abs)
- equalities (<>=)

Combining Forms:

- Math expressions
  - x
  - Real numbers
  - Mult of Expr * Expr
  - Sub of Expr * Expr
  - Div of Expr * Expr

  - Add of Expr * Expr
  - Sin of Expr
  - Cos of Expr
  - Sqrt of Expr
  - Abs of Expr
  - Pow of Expr * Expr

- Equations
  - $< y >< Eq >< Expr >$
- Bounds (default: canvas boundaries)
  - **R** $< y <$ **R**
  - **R** $< x <$ **R**
- Canvas $\rightarrow$ takes height and width
- Point $\rightarrow$ real number * real number
- Brush Style $\rightarrow$ List of points in 5 x 5 square
  - Real numbers must be between 0 and 5
- Drawing $\rightarrow$ Equation * Bounds * Color * Brush Style

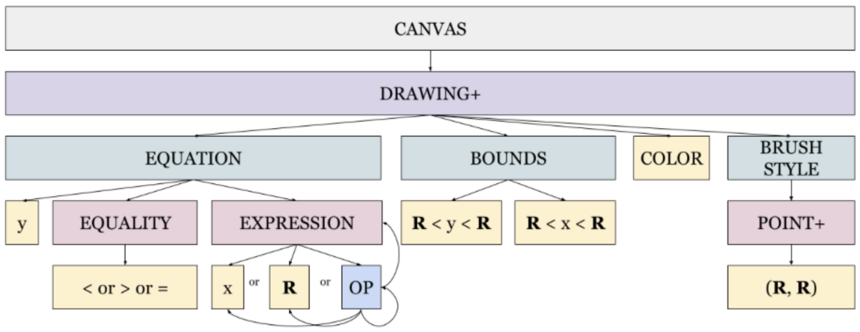
6 Semantics

i We are going to use the following primitive types in generating an expression: x, y, real numbers, operations and equality symbols. These will allow the users to generate the line of almost any equation they wish to draw. Color will also be a primitive type that can be used for shading and drawing lines with different colors.

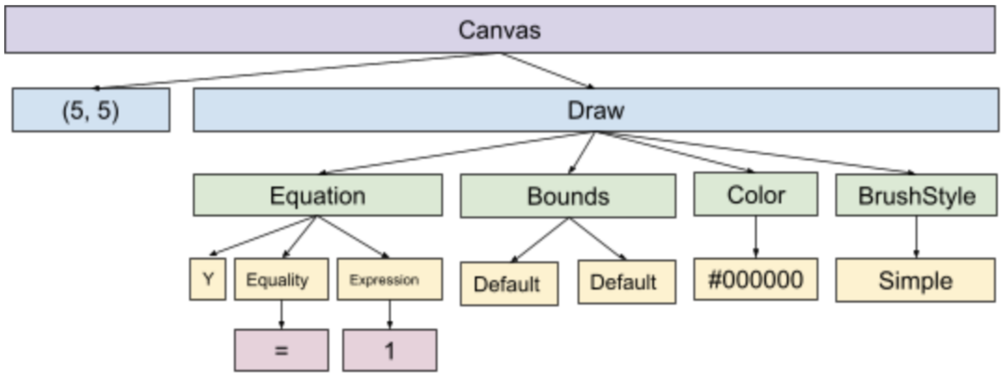
ii First, the user must create a canvas by inputting a height and width. Drawings will end at x and y unless other limits are specified. Drawings are the main elements in the language. They are a combination of an equation, its bounds, a color, and a brush style. Equations must have the y variable on the left, an $=, <$, or $>$, followed by a mathematical expression. Bounds denote the limits of the equation for x, y, or both. Mathematical expressions combine our operations, real numbers, and the x variable. Brush styles consist of a list of points, which we will connect to look like brush strokes.

Syntax	Abstract Syntax	Type	Prec./Assoc.	Meaning
n	Num of float	float	N/A	Primitive value, represent using the f# float type
$Y\|y$	Y	Y	N/A	Primitive value that we use when graphing equations
$X\|x$	X	Oper	N/A	Primitive value that we use when graphing equations
$<$	Less	Equality	N/A	Primitive value that we will use later when coloring shapes
$=$	Equal	Equality	N/A	Primitive value that we will use later when coloring shapes
$>$	Greater	Equality	N/A	Primitive value that we will use later when coloring shapes
$(o1 + o2)$	Add of Oper * Oper	Oper	Unambiguous (parens)	Combining form that we use when calculating the line to graph
$(o1 - o2)$	Sub of Oper * Oper	Oper	Unambiguous (parens)	Combining form that we use when calculating the line to graph
$(o1/o2)$	Div of Oper * Oper	Oper	Unambiguous (parens)	Combining form that we use when calculating the line to graph
$(o1 * o2)$	Mult of Oper * Oper	Oper	Unambiguous (parens)	Combining form that we use when calculating the line to graph
$(o1 \wedge o2)$	Pow of Oper $\wedge$ Oper	Oper	Unambiguous (parens)	Combining form that we use when calculating the line to graph
sin $o1$	Sin of Oper	Oper	Unambiguous (parens)	Combining form that we use when calculating the line to graph
cos $o1$	Cos of Oper	Oper	Unambiguous (parens)	Combining form that we use when calculating the line to graph
sqrt $o1$	Sqrt of Oper	Oper	Unambiguous (parens)	Combining form that we use when calculating the line to graph
abs $o1$	Abs of Oper	Oper	Unambiguous (parens)	Combining form that we use when calculating the line to graph

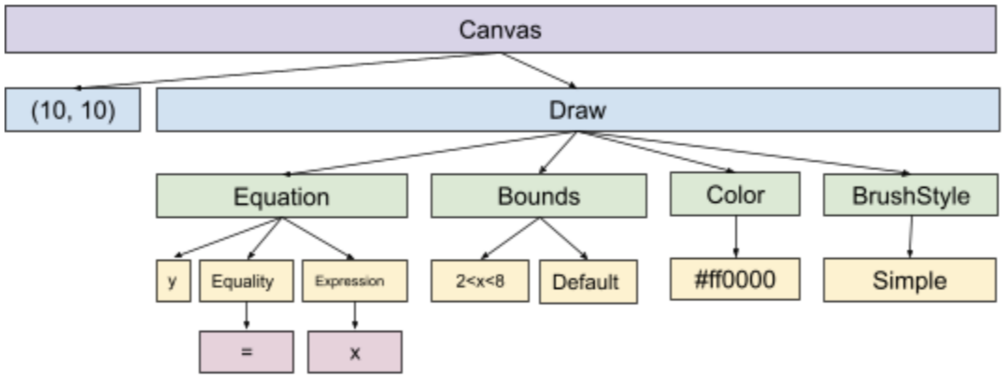
iii **Hierarchy Drawing:**



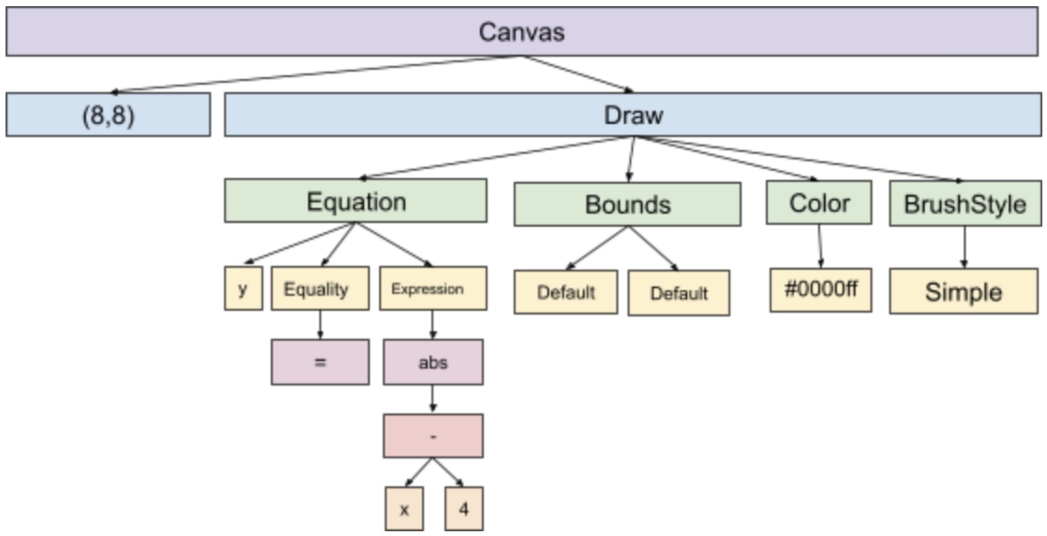
iv **AST for Example 1:**



AST for Example 2:



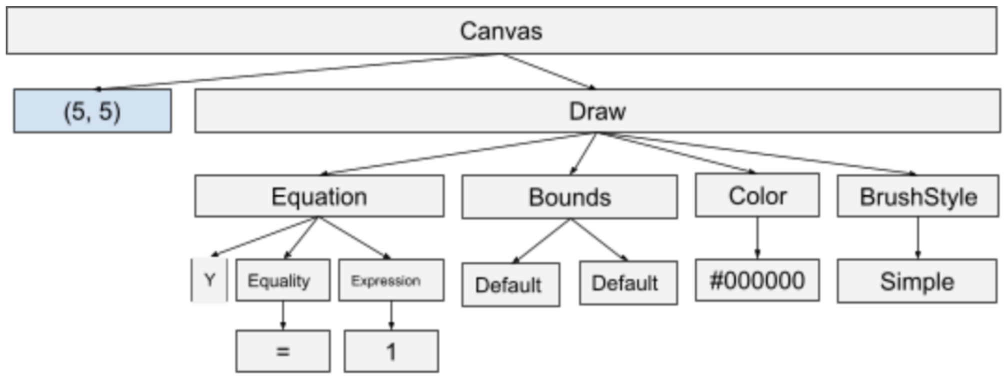
AST for Example 3:



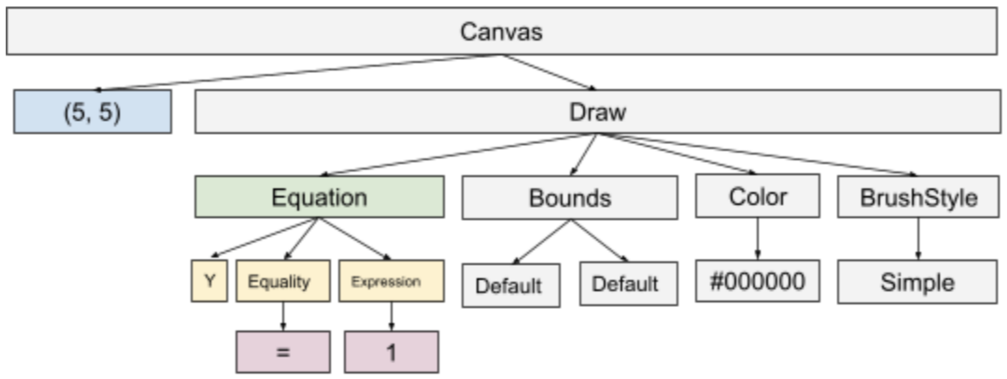
v No, our programs do not read any input. However, the user can input a program as a file so they do not have to retype all the commands.

vi The output of evaluating the program is a drawing consisting of the lines specified in the program. This will be in the form of an SVG file.

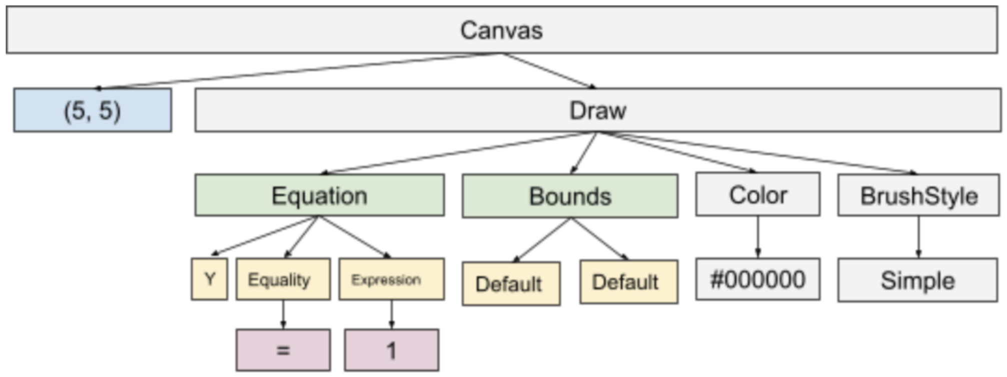
vii First, we get the canvas size.



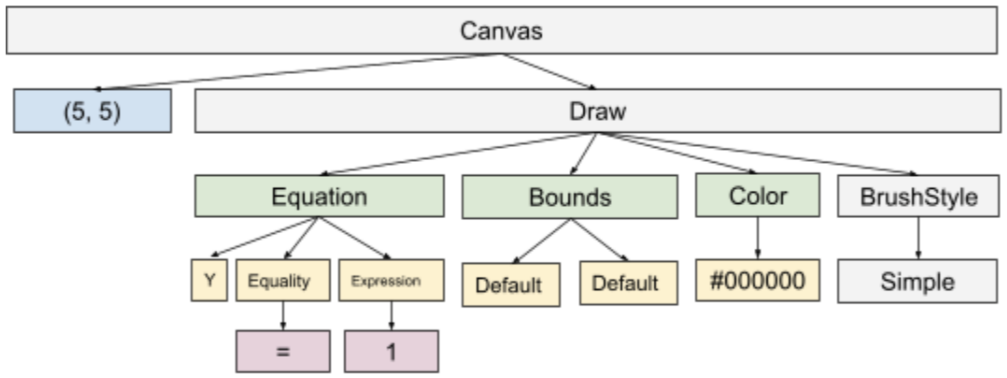
Then, we get the equation that we want to draw.



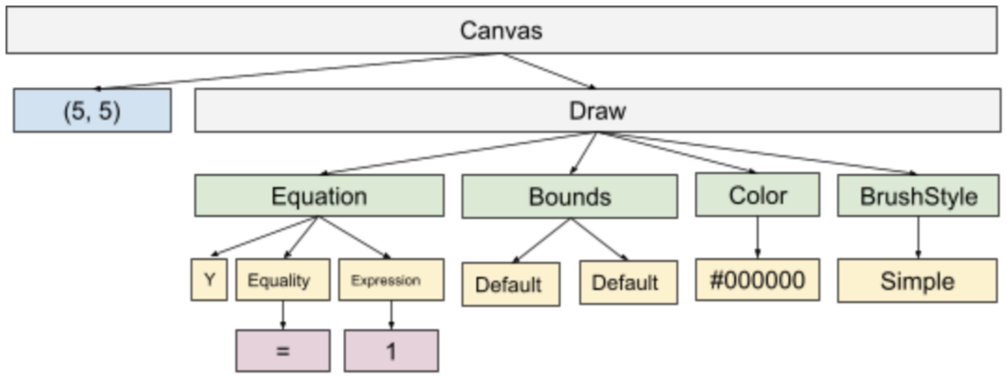
Next, we get the bounds for where to evaluate x and y. If they are default, they are the canvas size. So in this case, $0 <= x <= 5$ and $0 <= y <= 5$.



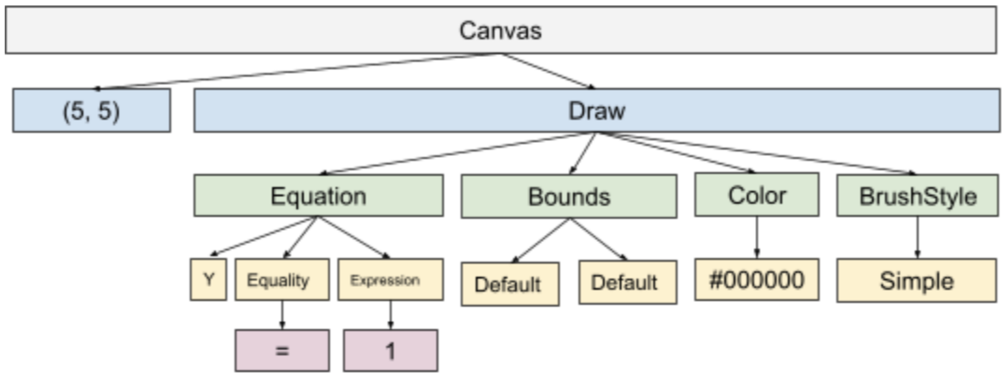
Then, we get the hex code for the color we would like to draw with. In this case it is #000000 for black.



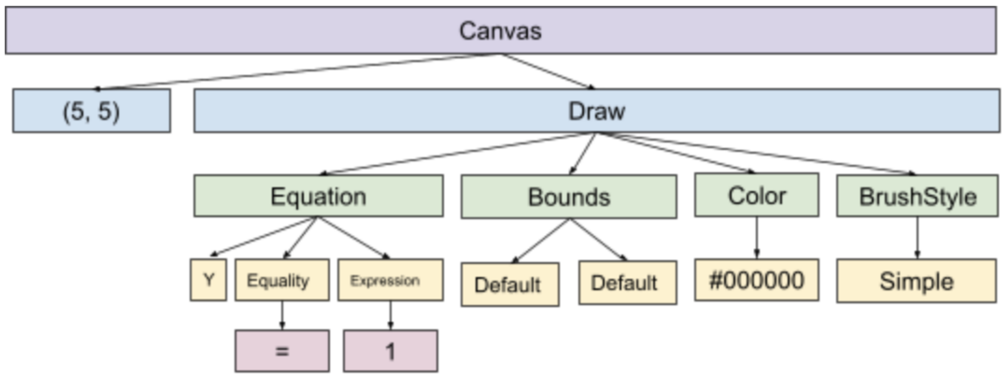
Then we get the brushStyle. In this case, it is ‘simple’ for just a single line.



Now, we have everything we need to draw the equation, so we do that using SVG.



We have no more draw commands left to evaluate, so we are done.



viii **Remaining Work:**
We would like to implement a custom brush feature, that allows the user to design a brush by entering a list of points.