

History Project Write-Up

James Gillespie

November 29, 2022

The idea behind my project is to demonstrate the compounding nature of Euclid's constructions. This is achieved through a web app I made called (I need a name). The nature of how I've made this is that, in programming a new construction, I am forced to use only Plato's Axioms and previously completed constructions. In fact, besides the Axioms, I am not allowed to program at all - instead, I constrain myself to writing a JSON formatted list of commands - each element is either a direct use of an Axiom, or a previous construction - to complete the construction.

The framework I made to allow for this style of "programming" uses JSON objects to store the commands for a construction. For example, the construction for a circle might look something like this (the exact format may have changed by now):

```
1  "name": "circle",
2      "fargs": [0,0,0,0],
3      "list": [
4          {
5              "jname": "point",
6              "args": [0, 0],
7              "farg": [0, 1],
8              "name": "P"
9          },
10         {
11             "jname": "point",
12             "args": [1,1],
13             "farg": [2, 3],
14             "name": "Q"
15         },
16         {
17             "jname": "circle",
18             "args": ["P", "Q"],
19             "farg": [],
20             "name": "A"
21         }
22     ],
```

The element "list" stores the commands needed for the construction. list[0] declares a point P located at either (0,0) or defined by arguments passed in elsewhere, using the "Axiom function" point(). list[1] similarly defines a point Q , and the "Axiom function" circle() is invoked to draw a circle centered at P , passing through Q .

Objects such as P and Q are stored in a list where they can be looked up by name - commands such as the call to "circle" require access to them. The circle itself will also be present in this list, allowing more complicated constructions to use "circle" as a command too.

Constructions can make use of other constructions by passing in the names of variables. If a construc-

tion it calls produces some object X it wants, it can bring X into its scope by setting the name field to $["X : A"]$, where A is the new name of the object.

Users can walk through each step of the process by pressing the back and forward buttons.

Flaws

- Some constructions do not lend themselves well to being automated. For example, proposition 12 requires placing a point below a line. The location of the point is totally arbitrary, except for that one requirement - and can easily be done by sight - but in code, it would have to be done algebraically, which would break the rules I set for myself.
- The way I pass arguments to complicated constructions is inadequate. If I want to make an equaliteral triangle (Prop I), with the base formed by two points that already exist, x and y , I have to insert those names into the Prop I construction. The issue with this is if x and y are already names used in Prop I, then everything falls apart - so for each construction I make, I limit myself to fewer and fewer names. This posed some serious problems later down the line, preventing me from making more complicated constructions.

Reflection

Overall, I feel pretty positive about this project. Although some of the technical details, such as designing the framework for reading instructions, and deploying the app to a web server, did take considerable time, the process writing up each construction was very informative. I'm also very pleased with the step-by-step feature.

Sources

My primary source has been Oliver Byrne's *Elements of Euclid*. I also used the resource

<http://aleph0.clarku.edu/~djoyce/elements/elements.html>