# Final Project Report

Priciples of Programming Languages

Bharat Sahlot — 2021111005

## 1 Introduction

After being confused about the goal of the project and TikZ syntax itself. I believe I have finally understood it(tiny bit of it) and broken it down to a few basic parts:-

- Commands (`\draw`, `\fill`, `\filldraw`, `\shade`, `\shadedraw`)
- Connectors (`circle, rectangle, curve, etc.`)
- Style (`draw, fill, minimum size, font, etc.`)
- Nodes

The final library looks a lot different than what was in Mid report and initial report. It is no longer a language, but a racket library.

## 2 Usage

`Ratikz` can be invoked using the `tikz` function. Its definition is as follows:

```
(define tikz
    (lambda commands
        ;; run the commands and,
        ;; enclose the output in tikz environment
        ;; return the generated string
        ))
```

You can pass multiple commands to it and each command can also be a list of commands. So you can for example put a `let` block inside `tikz` parameters.

A usage will look similar to

```
(tikz
    (draw (def-style)
          (list (p 0 0) (p 1 1))
          (list (edge)))
    (draw (def-style)
          (list (p 0 0) (p 1 0) (p 1 1))
          (list (edge) (edge) (edge cycle))))
```

### 2.1 Commands

In `Tikz` the commands are responsible for drawing anything, they all support options using styles. Below is example of 2 `Tikz` commands.

```
\node[circle, draw=green!60, ...more options] (lowercircle) [below=of maintopic] {4};
\draw[->] (uppercircle.south) --  (maintopic.north);
```

The base syntax is as follows,

```
\command[style] p1 <connector> p2 <connector> ...; % for drawing commands(draw, fill, fill
\node[style] (<name>) [<position>] {<text>}; % for node command
```

In Ratikz, the syntax for commands is as follows,

```
(<command> <style> (list <points>) (list <connectors>))

;; example
(draw (def-style) (list (p 2 2) (p 3 1)) (list (edge)))
;; results in \draw (2,2) -- (3,1);

(draw (def-style) (list (p 0 0) (p 1 0) (p 1 1)) (list (edge) (edge) (edge cycle)))
;; results in \draw (0,0) -- (1,0) -- (1,1) -- cycle;
```

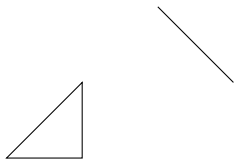

Figure 2.1: Figure generated from code

The drawing commands work as follows. Let $c_i$ be connector at index $i$ and $p_i$ be point at index $i$. Then, $c_i$ is placed between $p_i$ and $p_{i+1}$(if exists).

## 2.2 Connectors

Connectors are your way of defining how to connect/fill space between two points. The following connectors are currently supported by Ratikz,

- `circle (radius)`: draws a circle of radius `radius`
- `edge`: draws a straight line between points
- `rectangle`: draws a rectangle with point before as the bottom left and point after as top right.
- `node`: a node with text
- `curve (control-points)`: draws a bezier curve, with support for upto 2 control points
- `ellipse (r1 r2)`: draws a ellipse with radiuses r1 and r2
- `arc (s-angle e-angle radius)`: draws an arc
- `grid`: draws a grid, step size is specified using style

Each connector can also be grouped with other connectors, by passing the other connectors at the end of the connector call. So you can create a circle and a node at the same point.

```
(circle 2 (node "Intersection point" "west"))
```

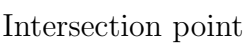


Figure 2.2: Figure generated from code

## 2.3 Style

`style` is just a class with some fields. It renders the fields in a format understable by `Tikz`.

## 2.4 Nodes

`node` isn't completely supported by `Ratikz` at the moment. Only the basic syntax is supported,

```
\node[style] (<name>) [<position>] {<text>};
```

Nodes are objects in `Ratikz`, i.e. they can be assigned to symbols and then used for positioning by other nodes.

```
(let
  ([maintopic (new node% (style squarednode) (name "maintopic") (text "2"))])
  (let ([uppercircle (new node% (style roundnode) (name "uppercircle")
                        (above maintopic) (text "1"))]
        [rightsquare (new node% (style squarednode) (name "rightsquare")
                        (right maintopic) (text "3"))]
        [lowercircle (new node% (style roundnode) (name "lowercircle")
                        (below maintopic) (text "4"))])
    (tikz
      maintopic uppercircle rightsquare lowercircle
      (draw arrow-style
            (list (a-p uppercircle "south") (a-p maintopic "north"))
            (list (edge)))
      (draw arrow-style
            (list (a-p maintopic "east") (a-p rightsquare "west"))
            (list (edge)))
      (draw arrow-style
            (list (a-p rightsquare "south") (a-p lowercircle "east"))
            (list (edge)))
      )))))
```

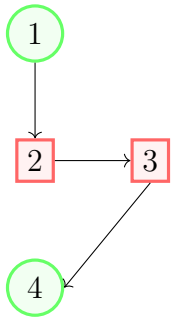


Figure 2.3: Figure generated from code

Positioning fields(`above, below, right, left, etc.`) can be assigned to position nodes with respect to other nodes.

## 2.5 Files

The code can be imported using the following:

```
(require "src/style.rkt")
(require "src/connectors.rkt")
(require "src/commands.rkt")
(require "src/point.rkt")
(require "src/ratikz.rkt")
```

# 3 Examples

## 3.1 Shapes

```
(display
  (let ([lst (unzip (map (lambda (i)
                           (list (p i i) (if (= i 2) (rectangle) (circle i))))
                         (inclusive-range 1 6)))])
    (let ([points (first lst)]
          [mpoints (map (lambda (x)
                          (p (- 12 (get-field x x)) (get-field y x))) (first lst))])
      (tikz
        (draw
          (new style% (draw "red") (fill "blue!50"))
          points (second lst))
        (draw
          (new style% (thickness "thick"))
          points (map (lambda (_) (edge)) (inclusive-range 1 5 2)))
        (draw
          (new style% (draw "blue") (fill "red!50"))
          mpoints (second lst))
        (draw (new style% (fill "yellow!50"))
              (list (p 4 0) (p 6 0) (p 5.7 2) cycle)
              (list (edge) (edge) (edge)))
        ))))
```

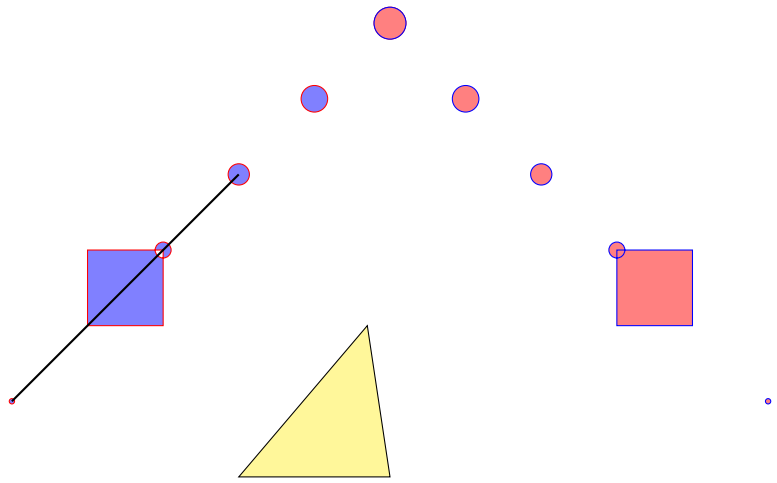

Figure 3.1: Shapes example

## 3.2 Curve

```
(display
  (let ([base-style (new style% (draw "gray"))])
    (tikz
      (draw base-style (list (p -2 0) (p 2 0)) (list (edge)))
      (filldraw base-style (list (p 0 0)) (list (circle 2)))
      (draw base-style (list (p -2 -2) (p 2 -2)) (list (curve (p 0 0))))
      (draw base-style (list (p -2 2) (p 2 2)) (list (curve (p -1 0) (p 1 0))))
      )))
```

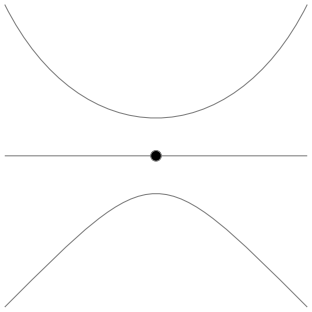

Figure 3.2: Curve example

## 3.3 Intersection

```
(display
  (let ([base-style (new style% (draw "gray") (thickness "thick"))])
    (tikz
      (draw base-style (list (p -1 2) (p 2 -4)) (list (edge)))
      (draw base-style (list (p -1 -1) (p 2 2)) (list (edge)))
      (filldraw (new style% (fill "black"))
                (list (p 0 0))
                (list (circle 2 (node "Intersection point" "west")))))))
```

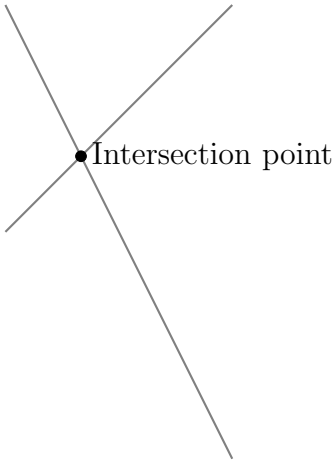


Figure 3.3: Intersection example

## 3.4 More Shapes

```
(display
  (let ([circle-style (new style%
                           (draw "red!60")
                           (fill "red!5")
                           (thickness "very thick"))])
    (tikz
      (filldraw circle-style
                (list (p -1 0))
                (list (circle 25)))
      (fill (new style%
                 (fill "blue!50"))
            (list (p 2.5 0))
            (list (ellipse 1.5 0.5)))
      (draw (new style%
```

```
            (thickness "ultra thick")
            (arrow "->"))
      (list (p 6.5 0))
      (list (arc 0 220 1)))
))
```

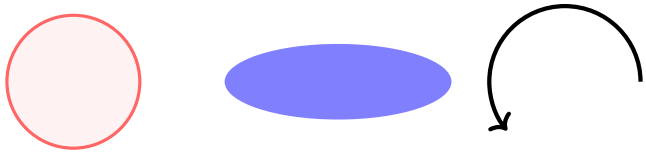

Figure 3.4: More shapes example

## 3.5 Grid

```
(display
  (let ([grid-style (new style%
                         (draw "gray")
                         (thickness "very thin")
                         (step 0.5))]
        [x-points (list -1 -0.5 0.5 1)]
        [y-points (list -1 -0.5 0.5 1)])
    (tikz
      (draw grid-style
            (list (p -1.4 -1.4) (p 1.4 1.4))
            (list (grid)))
      (draw (def-style)
            (list (p -1.5 0) (p 1.5 0))
            (list (edge)))
      (draw (def-style)
            (list (p 0 -1.5) (p 0 1.5))
            (list (edge)))
      (draw (def-style)
            (list (p 0 0))
            (list (circle 25)))
      (shadedraw (new style%
                      (left-color "gray")
                      (right-color "green")
                      (draw "green!50!black"))
            (list (p 0 0) (p 0.3 0))
            (list (edge) (arc 0 30 0.3) (edge cycle)))
      (map (lambda (x)
             (draw (new style% (font "\\tiny"))
                   (list (p x 0.1) (p x -0.1))
                   (list (edge (node (~a x) "north")))))
           x-points)
      (map (lambda (y)
             (draw (new style% (font "\\tiny"))
                   (list (p 0.1 y) (p -0.1 y))
                   (list (edge (node (~a y) "east")))))
           y-points)
       )))
```

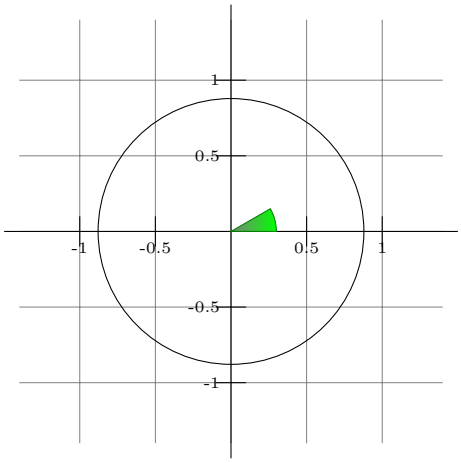


Figure 3.5: Grid example

# 4 Future Work

There is a lot that can be done to improve the experience:-

- Better cleaner syntax
- Support more connectors
- Units support
- Positioning library support
- Better coordinate specification eg. `++(down:3.5mm)`
- Math support while using nodes properties(*very interesting)*
- *A lot more can be done*