Rational Geometry with Racket

Ted Szylowiec

1 Developing the module

1.1 What's this about?

There are several software applications for interactive geometry. However, the one we are going to write ourselves serves to teach both geometry and computer programming. This simple geometry module, developed in Racket, can be used to study some surprisingly complicated and interesting two-dimensional problems. These problems are studied by typing interactively into the Racket REPL. The user sees the intermediate results of his reasoning and calculations. This assists learning and problem exploration. Besides that, I have found this geometry module to be an handy tool for designing problems, examples and examination questions.

From the perspective of computer science, this project gives the student an approachable introduction to software design. Rational Geometry with Racket was a one-semester topic taught to high school students. This article is a summary (with extensions and improvements) of what we did. What this article doesn't cover are the introductions to both the Racket language and to the mathematics of 2D analytic geometry. There is also a prerequisite of basic computer literacy: how to open and edit files, how to save your work, how to download and install applications, how to run something in the terminal. This latter prerequisite is not exactly trivial because (as I discovered) many students cannot meet it, despite years of studying computers in school. It's up to the teacher to cover all this too.

If you haven't installed Racket yet, go to `https://racket-lang.org`, download the system and install it. The installation is painlessly simple. You should be up and running in 5 minutes.

The rational geometry module begins the way most Racket modules begin: with some comments and `#lang racket`.

```
;; rational-geometry.rkt
;;
;; Geometry of points and lines using Racket.

#lang racket
```

Type this into a file called `rational-geometry.rkt` and save it. We will proceed to add code into this file and build our module.

1.2 Points and lines

The two fundamental geometrical objects that we are concerned with are two-dimensional points and two-dimensional lines. We will use the notation (x, y) for points and $[l, m, n]$ for lines, where the triple $[l, m, n]$ represents the line equation

$$lx + my + n = 0.$$

$A(3, 2)$ is shorthand, meaning that point A is at coordinate $(3, 2)$. A good way to represent points and lines in code is to use structs. Computation with structs is fast and efficient. Points and lines are things having internal structure and independent identity. Structs reflects this.

```
(struct point (x y) #:transparent)
(struct line (l m n) #:transparent)
```

When you create structs this way, a number of other functions are created for you. Accessor functions allow you to get the components inside the stuct.

```
> (define a (line 1 2 3))
> (line-m a)
2
> (define P (point 4 -7))
> (point-y P)
-7
```

Racket also creates predicate functions that test whether or not something is this type or that type of struct.

```
> (point? a)
#f
> (line? a)
#t
> (point? P)
#t
> (line? P)
#f
```

The predicate `equal?` compares structs in a simple element-by-element manner. Sometimes this is appropriate (it's fine for points), but other times it is not what we want (it's not fine for comparing lines.)

```
> (define Q (point 4 -7))
> (define R (point -3 8))
> (equal? Q P)
#t
> (equal? Q R)
#f
```

It's not that simple to compare lines because $[l, m, n]$ and $[kl, km, kn]$ are the same line. Lines are represented by equations

$$lx + my + n = 0$$

and these equations can be multiplied by any factor. In order to compare lines, this must be taken into account. Two lines $[l, m, n]$ and $[l', m', n']$ are equal if the following conditions hold:

$$\begin{aligned} lm' - ml' &= 0 \\ mn' - nm' &= 0 \\ ln' - nl' &= 0. \end{aligned}$$

In code,

```
(define (line-equal? line1 line2)
  (let ((l1 (line-l line1))
        (m1 (line-m line1))
        (n1 (line-n line1))
        (l2 (line-l line2))
        (m2 (line-m line2))
        (n2 (line-n line2)))
    (and (= 0 (det l1 m1 l2 m2))
         (= 0 (det m1 n1 m2 n2))
         (= 0 (det l1 n1 l2 n2)))))
```

Notice the repetetive code in the `let` expression. If we don't do something about this now, our geometry module will become full of code like that. To the rescue is Racket's pattern-matcher. Instead of picking apart components of points and lines using accessor functions, we do it all in one shot using `match-let`.

```
(define (line-equal? line1 line2)
  (match-let (((line l1 m1 n1) line1)
              ((line l2 m2 n2) line2))
    (and (= 0 (det l1 m1 l2 m2))
         (= 0 (det m1 n1 m2 n2))
         (= 0 (det l1 n1 l2 n2)))))
```

That looks much better.

1.3 Joining lines

Two lines can be joined to make a point. Cramer's method is a good way to find this intersection point. Besides that, Cramer's technique can be applied to countless other problems, so it is worth learning until a student can use it to solve linear equations with their eyes closed. It's also a very elegant idea and translates well into Racket code.

To solve this system:

$$\begin{aligned} lx + my + n &= 0 \\ l'x + m'y + n' &= 0 \end{aligned}$$

Compute the system determinant D

$$D = \begin{vmatrix} l & m \\ l' & m' \end{vmatrix}$$

and the x and y determinants

$$D_x = \begin{vmatrix} -n & m \\ -n' & m' \end{vmatrix}, \quad D_y = \begin{vmatrix} l & -n \\ l' & -n' \end{vmatrix}.$$

The solution x, y is

$$x = \frac{D_x}{D}, \quad y = \frac{D_y}{D}.$$

Determinant is defined as a seperate function so it can be used in many places in the module.

```
(define (det a b c d)
  (- (* a d)
     (* b c)))

(define (join-lines a b)
  (match-let (((line al am an) a)
              ((line bl bm bn) b))
    (let ((D (det al am bl bm))
          (Dx (det (- an) am (- bn) bm))
          (Dy (det al (- an) bl (- bn))))
      (point (/ Dx D)
             (/ Dy D)))))
```

1.4 Joining points

Two points define a line. If we begin with the canonical equation for a line,

$$lx + my + n = 0$$

we see that there are three variables to solve for: l, m, n, but only two points are given. One idea is to divide the line equation by n, giving an equation in two unknowns:

$$\frac{l}{n}x + \frac{m}{n}y + 1 = 0.$$

Set $p = l/n$, $q = m/n$. This equation must be satisfied at both of the given points $A(x_1, y_1)$, $B(x_2, y_2)$, which yields two equations in two unknowns:

$$\begin{aligned} px_1 + qy_1 + 1 = 0 \\ px_2 + qy_2 + 1 = 0. \end{aligned}$$

We can solve for p and q by Cramer's method:

$$D_p = \begin{vmatrix} -1 & y_1 \\ -1 & y_2 \end{vmatrix}, \quad D_q = \begin{vmatrix} x_1 & -1 \\ x_2 & -1 \end{vmatrix}, \quad D = \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}.$$

And now $p = D_p/D$, $q = D_q/D$ and our line is $[p, q, 1]$. But there is a problem with this method. What if the line goes through the origin? In this case, $n = 0$, and this method won't work due to division by zero. However, by examining

$$p = \frac{l}{n} = \frac{D_p}{D}, \quad q = \frac{m}{n} = \frac{D_q}{D}$$

it's clear that l, m, n can be identified with D_p, D_q and D. So we don't actually have to perform any divisions. Once the determinants are computed, we have the components of the line we are looking for:

$$[l, m, n] = [D_p, D_q, D].$$

```
(define (join-points A B)
  (match-let (((point x1 y1) A)
              ((point x2 y2) B))
    (let ((Dp (det -1 y1 -1 y2))
          (Dq (det x1 -1 x2 -1))
          (D (det x1 y1 x2 y2)))
      (line Dp Dq D))))
```

1.5 Prettification

As we will see, computations in rational geometry often leads to formidable fractions. We would prefer to avoid them if we can. For points, we have no choice but to live with big fractions, unless we resort to homogeneous coordinates. But in the case of lines, we can improve their appearance. This is because a line tuple $[l, m, n]$ can be multiplied by any factor and remain the same line. Denominators of fractions can be eliminated in lines.

```
(define (eliminate-fractions myline)
  (match-let (((line l m n) myline))
    (let ((dl (denominator l))
          (dm (denominator m))
          (dn (denominator n)))
      (let ((d (* dl dm dn)))
        (line (* d l)
              (* d m)
              (* d n))))))
```

Common factors should be eliminated wherever possible. There's no need to carry around $[44, 88, 33]$ when $[1, 8, 3]$ will do.

```
(define (eliminate-factors myline)
  (match-let (((line l m n) myline))
    (let ((g (gcd l m n)))
      (line (/ l g) (/ m g) (/ n g)))))
```

Likewise, eliminate excess negative numbers from lines. Better to use $[1, 1, 1]$ instead of $[-1, -1, -1]$.

```
(define (neg1 x)
  (if (< x 0) 1 0))

(define (count-negatives l m n)
  (+ (neg1 l)
     (neg1 m)
     (neg1 n)))

(define (fix-negatives myline)
  (match-let (((line l m n) myline))
    (if (< (count-negatives l m n) 2)
        myline
        (line (- l) (- m) (- n)))))
```

These cosmetic effects are not really necessary. But they do make the end user's experience a little better. The complete prettification function is built from the smaller components defined above.

```
(define (prettify myline)
  (fix-negatives
   (eliminate-factors
    (eliminate-fractions myline))))
```

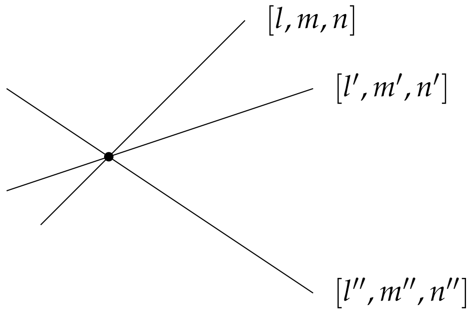
The function `join-points` does its job but does not prettify its results. To make things convenient for the user, we create a universal `join` function that handles both points *and* lines, and prettifies its result when returning a line. This is accomplished by using the predicates `point?` and `line?` that Racket created for us.

```
(define (join U V)
  (cond ((and (point? U)
              (point? V))
         (prettify (join-points U V)))
        ((and (line? U)
              (line? V))
         (join-lines U V))
        (else
          (error
            'join
            "Cannot join arguments."))))
```

This function is the backbone of the rational geometry module.

1.6 Concurrent and collinear

Three lines are concurrent when they are on the same point.



If (x,y) are the coordinates of the point of concurrency, then the following equations must hold:

$$\begin{aligned} lx &+ my &+ n &= 0 \\ l'x &+ m'y &+ n' &= 0 \\ l''x &+ m''y &+ n'' &= 0 \end{aligned}$$

Since we have three equations in only two unknowns x and y, it must be that one of these equations is a linear combination of the other two. This happens when the determinant of the coefficients is zero:

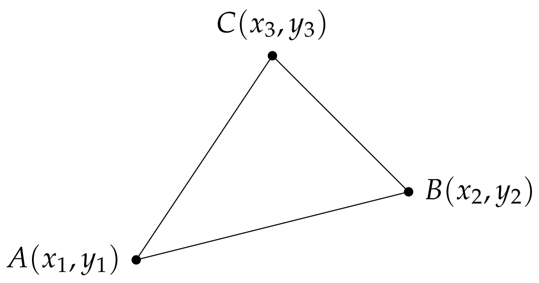
$$\begin{vmatrix} l & m & n \\ l' & m' & n' \\ l'' & m'' & n'' \end{vmatrix} = 0.$$

This is the condition for three lines to be concurrent. We define a seperate `det3` function so that it can be used in other contexts.

```
(define (det3 a b c d e f g h i)
  (+ (* a (det e f h i))
     (* -1 b (det d f g i))
     (* c (det d e g h))))
```

```
(define (concurrent? a b c)
  (match-let (((line al am an) a)
              ((line bl bm bn) b)
              ((line cl cm cn) c))
    (= 0 (det3 al am an
               bl bm bn
               cl cm cn))))
```

A condition for collinearity of points can be discovered by examining the area of a triangle. Consider triangle ABC.



The are S of ABC can be computed by the remarkable formula

$$S = \frac{1}{2}\left(\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} + \begin{vmatrix} x_3 & y_3 \\ x_1 & y_1 \end{vmatrix}\right).$$

If the verticies are taken in the counter-clockwise direction then S will be positive. If clockwise, S will be negative. This formula is worth knowing because it is easily remembered and it can also be extended to polygons with more verticies. The `det` function is reused to write `triangle-area`.

```
(define (triangle-area A B C)
  (match-let (((point Ax Ay) A)
              ((point Bx By) B)
              ((point Cx Cy) C))
    (* 1/2
       (+ (det Ax Ay Bx By)
          (det Bx By Cx Cy)
          (det Cx Cy Ax Ay)))))
```

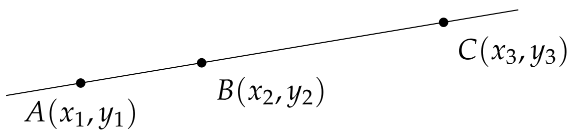
Now let's use the idea of triangle area to derive a condition for collinearity of points. By using the determinant property

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = -\begin{vmatrix} c & d \\ a & b \end{vmatrix}$$

it is easy to verify that the S formula is equivalent to the following 3-by-3 determinant:

$$S = \frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}.$$

Three points are collinear when they are on the same line.



But if three points were not collinear, then they would form a triangle with nonzero area. So intuitively we arrive at the condition that three points are collinear if

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0$$

Written in terms of 3-by-3 determinants, the conditions for concurrency of lines and collinearity of points are both beautiful and easy to remember. Reusing `det3`, we have:

```
(define (collinear? A B C)
  (match-let (((point Ax Ay) A)
              ((point Bx By) B)
              ((point Cx Cy) C))
    (= 0 (det3 Ax Ay 1
               Bx By 1
               Cx Cy 1))))
```

2 Adding features

We are finished the basic functionality of the rational geometry module. As it stands, it can be used to study non-metrical or projective types of geometry problems. But we can make it more powerful and more user-friendly by adding some features.

It is often necessary to compute the slope of a line. This can be done from scratch every time by using accessor functions. But why not simply build this common task into the geometry module? While we are at it, it is very handy to have a function that computes the slope perpendicular to a given line. We call this `anti-slope`.

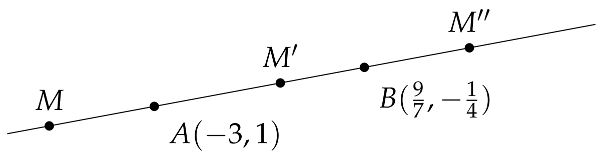
```
(define (slope a)
  (let ((al (line-l a))
        (am (line-m a)))
    (- (/ al am))))

(define (anti-slope a)
  (- (/ (slope a))))
```

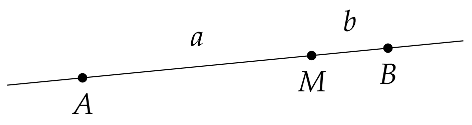
Other features that we should have: ways of computing ranges and pencils, and a procedure to compute that most interesting geometrical quantity: the *anharmonic ratio.*

2.1 Ranges and pencils

A common task in geometry is to construct a range of points on a line. The line can be defined by two base points A and B. Given A and B, we would like to generate points M, M', M'' wherever we need them.



One method is *proportional division*. Given proportions a, b, where must we place M such that segment AB is divided according to the proportions $AM : MB = a : b$?



Using the notation of boldface letters to denote point tuples, the coordinates of point M can be expressed as a linear combination of the coordinates of A and B:

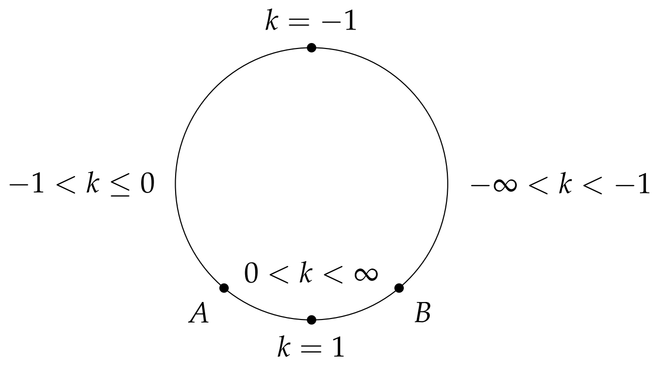
$$\boldsymbol{M} = \frac{b\boldsymbol{A} + a\boldsymbol{B}}{b+a}.$$

Where $\boldsymbol{M} = (M_x, M_y)$ and so on. Dividing through by b and substituting $k = a/b$ we obtain the segment division formula:

$$\boldsymbol{M} = \frac{\boldsymbol{A} + k\boldsymbol{B}}{1+k}.$$

If you experiment with this formula you will quickly see that when k is positive, the point M will be in between A and B, and when k is negative, M will lie outside the segment AB.

At first it may be hard to see exactly how k parametrizes the position of M. It helps to imagine line AB as actually being a circle of infinte radius, having regions where k is positive or negative.



The point corresponding to $k = 1$ is the midpoint of AB, while the point corresponding to $k = -1$ is the *exterior* midpoint of AB, which is the point at infinity. Points corresponding to positive values of k lie in between A and B, while points corresponding to negative k lie to the left of A or to the right of B.

We will use a *higher-order function* to build ranges. A higher-order function is a function that returns another function. This is quite useful in practice because it allows us to make a range function once from base points A and B, and then use this function many times to make however many points M, M', M'' we need by passing in values of k.

```
(define (make-range A B)
  (match-let (((point Ax Ay) A)
              ((point Bx By) B))
    (lambda (k)
    (let ((a (+ 1 k)))
      (point (/ (+ Ax (* k Bx)) a)
             (/ (+ Ay (* k By)) a))))))
```

That's all there is to it. For a demonstration, we create the points M, M', M'' given A, B as in the earlier figure. Define the base points A, B:

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define A (point -3 1))
> (define B (point 9/7 -1/4))
```

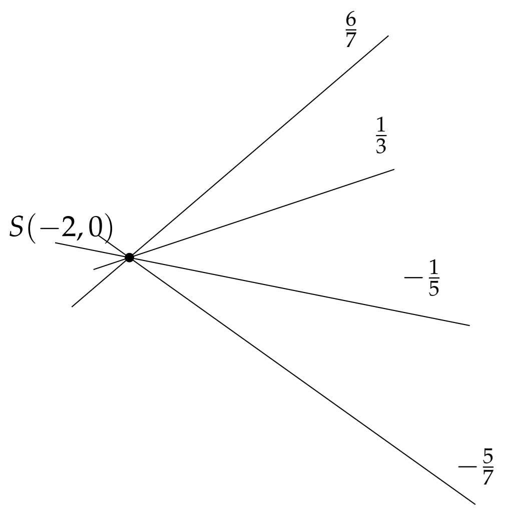
Create the range function and define some division ratios.

```
> (define r (make-range A B))
> (define k -1/3)
> (define k1 3/2)
> (define k11 -3)
```

Construct the required points.

```
> (define M (r k))
> (define M1 (r k1))
> (define M11 (r k2))
> (values M M1 M11)
(point -36/7 13/8)
(point -3/7 1/4)
(point 24/7 -7/8)
```

We know how to construct ranges, which are many points on a line, what about constructing many lines on a point? These are called *pencils*.



It's possible to construct pencils in a way analogous to what we did for ranges. But if so, k no longer has a simple interpretation in terms of ratios or proportions. Instead, we will choose lines from a pencil by specifying their slopes. Suppose the pencil center is at (x_0, y_0). From the equation of slope:

$$\frac{y - x_0}{x - x_0} = k.$$

Writing this in the canonical line form gives

$$-kx + y + kx_0 - y_0 = 0$$

or

$$[-k, 1, kx_0 - y_0].$$

As with `make-range`, the `make-pencil` procedure returns a function of k. But this time k is the slope of the desired line.

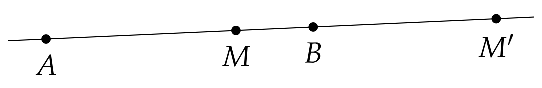
```
(define (make-pencil M)
  (match-let (((point Mx My) M))
    (lambda (k)
      (prettify (line (- k)
                      1
                      (- (* k Mx)
                         My))))))
```

Here is how you would construct the lines in the figure.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define S (point -2 0))
> (define penS (make-pencil S))
> (define a (penS 6/7))
> (define b (penS 1/3))
> (define c (penS -1/5))
> (define d (penS -5/7))
> (values a b c d)
(line 6 -7 12)
(line 1 -3 2)
(line 1 5 2)
(line 5 7 10)
```

2.2 The anharmonic ratio

Consider two base points A and B as in the figure below. What if there are *two* points of division M and M'? Then there would be two division ratios: one corresponding to M and the other corresponding to M'. Call these ratios k and k'. It doesn't matter if M and M' are inside or outside the segment AB.



A natural thing to do is to form the quantity k/k'. It turns out that this quantity has almost magical properties. Entire books have been written about it. It goes by the name *anharmonic ratio* or *cross-ratio* and the usual symbol given to it is $(ABMM')$.

Expressing k and k' in terms of segment proportions gives the definition for anharmonic ratio:

$$(ABMM') = \frac{k}{k'} = \frac{AM/MB}{AM'/M'B}.$$

Since it is easy to remember the definition for division ratio k, it is also easy to remember this definition of anharmonic ratio. Division ratio can be obtained by solving

$$\boldsymbol{M} = \frac{\boldsymbol{A} + k\boldsymbol{B}}{1 + k}$$

for k. In practice we only need either the x-coordinates of these points, or the y-coordinates. We choose the x-coordinates for our code. In doing so we hope that the points A, B, M do not lie on a vertcal line! In Racket code, anharmonic-ratio is built from division-ratio. Both are useful functions, so it makes sense to separate them.

```
(define (division-ratio A B M)
  (if (collinear? A B M)
      (let ((Mx (point-x M))
            (Ax (point-x A))
            (Bx (point-x B)))
        (/ (- Mx Ax) (- Bx Mx)))
      (error 'division-ratio
             "A B M must be collinear")))

(define (anharmonic-ratio A B M M1)
  (/ (division-ratio A B M)
     (division-ratio A B M1)))
```

2.3 Packaging the library

What functions and symbols from our module should be made available to the user? Surely there are things in rational-geometry that the user does not need to see, such as the functions joint-points and join-lines. We want the user to use the universal join function instead. Controlling what the user has access to in our module is done with provide:

```
(provide ...
         ...
         ...)
```

If we want the user to use something defined in the module, we put it in provide. Certainly the user needs point and line structs, so these belong in provide. But the user also needs all the functions associated with point and line structs, such as the accessors point-x, line-l, etc., and the predicates point? and line?. Racket has a simple mechanism, struct-out for providing everything associated with a struct:

```
(provide (struct-out point)
         (struct-out line))
```

A function like det3 is useful on its own, so we definitely want to provide it to the user. What we don't want to provide are the point and line joining functions. We want the user to use only join, which, in addition to being more convenient, prettifies its results. Also, experimental functions or things that may have been defined for the purpose of testing should not be provided. Those details are best kept away from the users of our geometry module.

```
(provide (struct-out point)
         (struct-out line)
         line-equal?
         det
         det3
         prettify
         join
         concurrent?
         collinear?
         triangle-area
         slope
         anti-slope
         make-pencil
         make-range
         division-ratio
         anharmonic-ratio)
```

The geometry module is ready to use.

3 Problem solving in the REPL

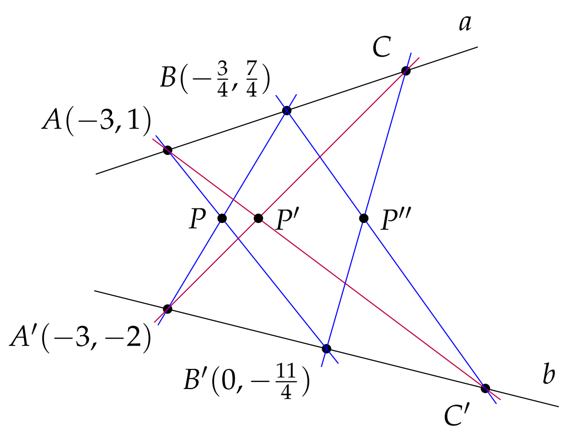
REPL stands for *read-eval-print loop.* This is where the user interacts with Racket. The three most popular ways to use the Racket REPL is through DrRacket, Emacs, or the terminal. For this project, we will simply use the terminal.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
>
```

We are now in the REPL and all our provided module functions are ready to be used. To quit the REPL, use <ctrl-D>.

3.1 Pappus points are collinear

Given two lines a and b, choose any three points A, B, C on line a, and any three points A', B', C' on line b. These points can be joined in a special way to define three points P, P' and P'' which we call *Pappus points*, after the ancient Greek mathematician. Pappus points are very interesting because they always lie along a straight line.



We are given A, B and A', B'. Choose points C and C', determine the resulting Pappus points, and show that they are collinear.

Choosing C and C' is easy to do with `make-range`. To place C to the right of segment AB, choose k in the interval $-\infty < k < -1$. It's convenient to choose $k = -2$.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define A (point -3 1))
> (define B (point -3/4 7/4))
> (define r (make-range A B))
> (r -2)
(point 3/2 5/2)
> (define C (r -2))
```

Do the same for C'.

```
> (define A1 (point -3 -2))
> (define B1 (point 0 -11/4))
> (define r1 (make-range A1 B1))
> (r1 -2)
(point 3 -7/2)
> (define C1 (r1 -2))
```

Construct the Pappus points by joining pairs of points into lines, then joining pairs of lines into points.

```
> (define P (join (join A B1)
                  (join A1 B)))
> (define P1 (join (join A C1)
                   (join A1 C)))
> (define P11 (join (join B C1)
                    (join B1 C)))
```

Let's have a look at these Pappus points.

```
> (values P P1 P11)
(point -69/35 -2/7)
(point -9/7 -2/7)
(point 69/98 -2/7)
```

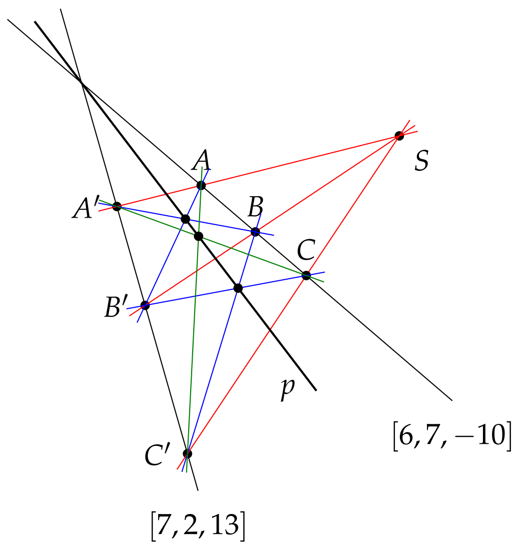
Do P, P', P'' lie on a straight line?

```
> (collinear? P P1 P11)
#t
```

Yes!

3.2 Perspective Pappus line is concurrent

Something even more interesting happens when the points A, B, C and A', B', C' are in perspective. In this case, the line p through the Pappus points P, P', P'' will be concurrent with the two given lines.



The given lines are $[7,2,13]$ and $[6,7,-10]$. Choose a center of perspective S and construct points on these lines that are related by perspective, as shown in the figure. Construct Pappus points and the Pappus line p. Verify that the resulting Pappus line is concurrent with the two given lines.

Begin by defining the given lines and choosing a place to put S, which can be anywhere.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define m (line 7 2 13))
> (define n (line 6 7 -10))
> (define S (point 3 3))
```

A pencil of lines on S determines the perspective relationship of points on the given lines. We can choose suitable perspective lines from the pencil on S by specifying their slopes.

```
> (define penS (make-pencil S))
> (define perspective1 (penS 1/4))
> (define perspective2 (penS 2/3))
> (define perspective3 (penS 3/2))
```

Use these perspective lines to find points A, B, C and A', B', C':

```
> (define A (join perspective1 n))
> (define B (join perspective2 n))
> (define C (join perspective3 n))
> (define A1 (join perspective1 m))
> (define B1 (join perspective2 m))
> (define C1 (join perspective3 m))
```

Let's have a look at these points.

```
> (values A B C)
(point -23/31 64/31)
(point 9/32 19/16)
(point 41/33 4/11)
> (values A1 B1 C1)
(point -7/3 5/3)
(point -9/5 -1/5)
(point -1 -3)
```

Find the Pappus points by joining pairs of points into lines and then joining pairs of lines.

```
> (define P (join (join A1 B)
                  (join A B1)))
> (define P1 (join (join C1 A)
                   (join C A1)))
> (define P11 (join (join C1 B)
                    (join B1 C)))
> (values P P1 P11)
(point -537/517 739/517)
(point -121/153 169/153)
(point -15/343 43/343)
```

The Pappus points should be collinear.

```
> (collinear? P P1 P11)
#t
```

The Pappus line is the join of any two of these points. Verify that the Pappus line is concurrent with the given lines.

```
> (define p (join P P11))
> p
(line 443 338 -23)
> (concurrent? m p n)
#t
```

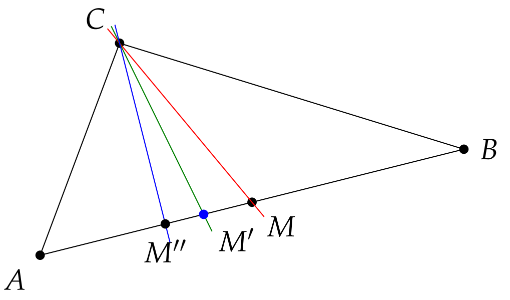
Incidentally, what is the point of concurrency?

```
> (join m n)
(point -3 4)
```

We are done.

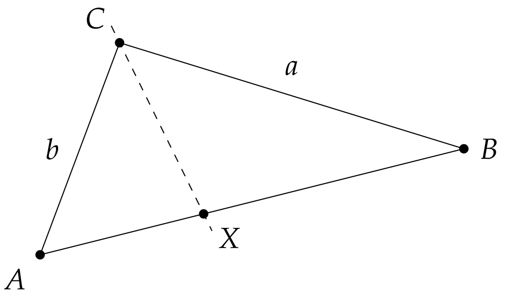
3.3 Altitude, angle bisector, median

Consider three lines connected with triangles: the angle bisector, the median line, and the altitude line. Let M be the point where the median line from A cuts side BC. Let M' be where the angle bisector at A cuts the side BC, and let M'' be the point where the perpendicular altitude from A cuts BC.



Here is an interesting phenomenon: the angle bisector point M' is always in between the altitude point M'' and the median point M.

Using the functions in our geometry module, how would we compute angle bisectors? Consider triangle ABC in the following figure and let a and b be the side lengths $\overline{CB}$ and $\overline{AC}$ respectively.



It can be shown that $\angle ACX$ and $\angle XCB$ are equal precisely when the proportion $AX : XB$ equals the proportion $b : a$. In other words, point X can be computed by proportional division

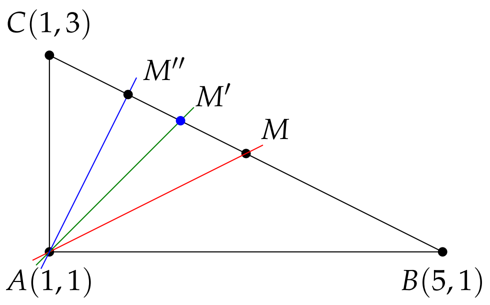
$$\boldsymbol{X} = \frac{\boldsymbol{A} + k\boldsymbol{B}}{1 + k}$$

by setting

$$k = \frac{AX}{XB} = \frac{b}{a}.$$

Reflecting on this for a moment, it becomes clear that, for the case of a general triangle, we cannot compute X this way with the tools we presently have in our module, because we have no way of computing distance. It's not difficult to add a function that computes distance, but this will introduce irrational numbers like $\sqrt{2}$. It is not clear how we should deal with them at this point.

However, it is possible to examine specific instances of this phenomenon where the side lengths a and b happen to be computable without the distance formula. This is the case when angle ACB is a right angle, and sides CA, CB are aligned parallel to the x and y axes:



Given A, B, C as above, compute the median line, the altitude line and the angle bisector line. Verify that the angle bisector point is in between the median and altitude points.

Begin by defining the given points in the Racket REPL.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define A (point 1 1))
> (define B (point 5 1))
> (define C (point 1 3))
```

The median line at A goes from A to the midpoint of CB. Midpoints can be computed using `make-range` with k set to 1.

```
> (define r (make-range C B))
> (define M (r 1))
> M
(point 3 2)
> (define median-line (join A M))
> median-line
(line 1 -2 1)
```

Our median line is $[1, -2, 1]$. To construct an altitude line from point A, first define a pencil of lines on A. Then choose the line which has a slope perpendicular to line CB. To do that, we use the `anti-slope` function.

```
> (define penA (make-pencil A))
> (define CB (join C B))
> (define altitude-line
    (penA (anti-slope CB))
```

Let's pause for a moment to check that the altitude line really is perpendicular to line CB. If the slope of the altitude line is k, then the slope of CB must be $-1/k$.

```
> altitude-line
  (line -2 1 1)
> (slope altitude-line)
2
> (slope CB)
-1/2
```

As we expect. Now for the altitude point M'':

```
> (define M11 (join altitude-line CB))
> M11
(point 9/5 13/5)
```

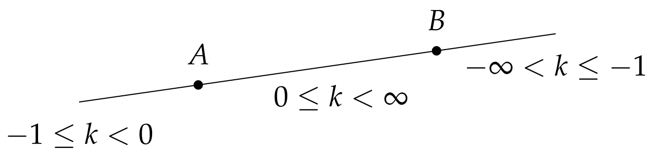
The lengths of the sides AB and AC can be easily read off the diagram for this particular triangle. Therefore we can apply the technique of proportional division, as explained above, to find the angle bisector line. The side AC has length 2 and the side AB has length 4. So the proportion that defines M' is 2 : 4 along segment CB.

```
> (define r (make-range C B))
> (define M1 (r 2/4))
> M1
(point 7/3 7/3)
```

Points M, M', M'' should be collinear.

```
> (collinear? M M1 M11)
#t
```

Finally, we want to show that M' is in between M'' and M. How can we do that? Consider two points A and B another point M that divides AB proportionally according to some ratio k. If k is negative, then M lies either to the left or to the right of segment AB. But if k is positive, then M is in between A and B.



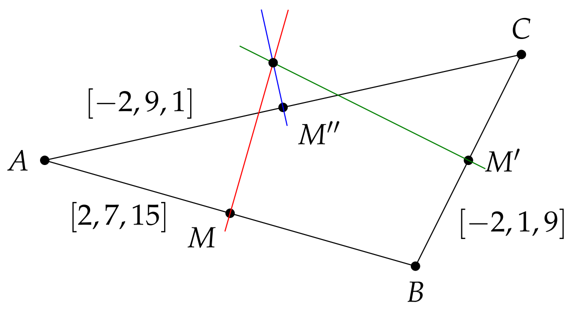
The module function `division-ratio` finds k given A, B and M. If `(division-ratio A B M)` is positive, then M is in between A and B. It remains to carefully place the altitude point, the median point and the angle bisector point into the proper argument positions for `division-ratio`:

```
> (division-ratio M11 M M1)
4/5
```

k is positive, therefore M' is in between M'' and M.

3.4 Side bisectors are concurrent

For the following triangle, demonstrate that the side bisectors are concurrent.



In this problem, we are given the lines that form the sides of the triangle. We see from this figure that sometimes the point of concurrency can be outside the triangle. Our strategy will be to calculate the midpoints of each side, then construct perpendicular lines on the midpoints using pencils. Begin by defining the lines that make up the sides of the triangle and use `join` to find the verticies.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define AC (line -2 9 1))
> (define AB (line 2 7 15))
> (define BC (line -2 1 9))
> (define A (join AC AB))
> (define B (join AB BC))
> (define C (join AC BC))
> (values A B C)
(point -4 -1)
(point 3 -3)
(point 5 1)
```

Now find the midpoints. The expression `(make-range X Y)` is actually a function because `make-range` returns functions of k. Therefore we can call the result of this expression without using any intermediate definitions as if it was a function, because it is, in fact, a function. This idea is called *anonymous functions*. We are using functions created by `make-range` without giving them explicit names.

```
> (define M ((make-range A B) 1))
> (define M1 ((make-range B C) 1))
> (define M11 ((make-range A C) 1))
> (values M M1 M11)
(point -1/2 -2)
(point 4 -1)
(point 1/2 0)
```

Likewise `make-pencil` returns functions of slope. If we want to, we can call these functions anonymously without using intermediate definitions.

```
> (define b
    ((make-pencil M)
       (anti-slope (join A B))))
> (define b1
    ((make-pencil M1)
       (anti-slope (join B C))))
> (define b11
    ((make-pencil M11)
       (anti-slope (join A C))))
> (values b b1 b11)
(line -14 4 1)
(line 1 2 -2)
(line 18 4 -9)
```

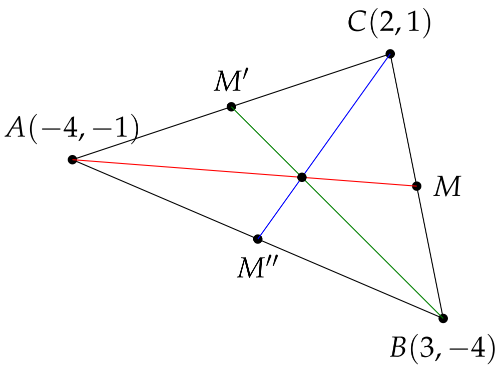
Are the perpendicular bisector lines b, b', b'' concurrent?

```
> (concurrent? b b1 b11)
#t
```

Yes!

3.5 Medians are concurrent

Not only are median lines always concurrent, there is an interesting surprise waiting at the end of this problem. We want to use our geomentry module to verify that median lines are concurrent for the following triangle.



Begin by defining symbols for the given points.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define A (point -4 -1))
> (define B (point 3 -4))
> (define C (point 2 1))
```

To compute midpoints, we will again call the functions created by `make-range` anonymously as we did in the previous problem.

```
> (define M ((make-range B C) 1))
> (define M1 ((make-range A C) 1))
> (define M11 ((make-range A B) 1))
> (values M M1 M11)
(point 5/2 -3/2)
(point -1 0)
(point -1/2 -5/2)
```

Now construct the median lines and verify that they are concurrent.

```
> (define AM (join A M))
> (define BM1 (join B M1))
> (define CM11 (join C M11))
> (values AM BM1 CM11)
(line 1 13 17)
(line 1 1 1)
(line -7 5 9)
> (concurrent? AM BM1 CM11)
#t
```

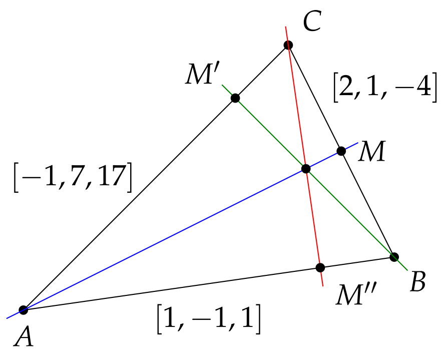
As expected. But examine the median lines in the figure. They all seem similar. The point of concurrency, X, seems to divide each median line in a similar way. We can make this observation more precise by using `division-ratio`.

```
> (define X (join AM BM1))
> X
(point 1/3 -4/3)
> (division-ratio A M X)
2
> (division-ratio B M1 X)
2
> (division-ratio C M11 X)
2
```

Interesting! The point of concurrency X divides each median line in a $2:1$ proportion. Another way to say this is that X trisects each median line. It turns out this isn't merely true for our particular triangle. It can be proved in general for all triangles using the methods of elementary geometry.

3.6 Altitudes are concurrent

The sides of the triangle are given. Verify that the altitudes are concurrent.



It turns out that we don't actually need to find the points M, M', M''. All we need are the verticies of the triangle. The concurrency of the altitude lines can then be elegantly demonstrated by using `make-pencil` and `anti-slope`.

Define symbols for the given lines and find the verticies.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define AC (line -1 7 17))
> (define BC (line 2 1 -4))
> (define AB (line 1 -1 1))
> (define A (join AC AB))
> (define B (join AB BC))
> (define C (join AC BC))
> (values A B C)
(point -4 -3)
(point 1 2)
(point 3 -2)
```

Create pencils on each vertex, and from each pencil select the line that is perpendicular to the opposite side. This time we will give explicit names to the functions created by `make-pencil`.

```
> (define penA (make-pencil A))
> (define penB (make-pencil B))
> (define penC (make-pencil C))
> (define a (penA (anti-slope BC)))
> (define a1 (penB (anti-slope AC)))
> (define a11 (penC (anti-slope AB)))
> (values a a1 a11)
(line -1 2 2)
(line 7 1 -9)
(line 1 1 -1)
```

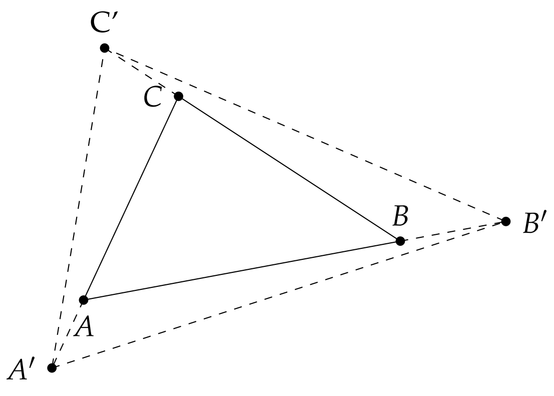
Are the altitude lines a, a', a'' concurrent?

```
> (concurrent? a a1 a11)
#t
```

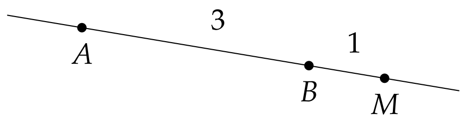
Of course they are!

3.7 Area of extended triangle

Consider the triange ABC in the figure below. Extend each side in the counterclockwise direction by $1/3$, making the larger triangle $A'B'C'$. We are given $A(-2,4)$, $B(1,-1)$ and $C(4,5)$. What is the ratio of the areas of the two triangles?



The extension points A', B', C' can be found by proportional division, but we must be a bit careful.



Here M extends segment AB by a third. The proportions are $AM : MB = 4 : -1$, giving a division ratio of $k = AM/MB = -4$. The triangle sides in the figure can be extended in the required manner by using `make-range` with $k = -4$.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define A (point -2 4))
> (define B (point 1 -1))
> (define C (point 4 -5))
> (define A1 ((make-range C A) -4))
> (define B1 ((make-range A B) -4))
> (define C1 ((make-range B C) -4))
> (values A1 B1 C1)
(point -4 7)
(point 2 -8/3)
(point 5 -19/3)
```

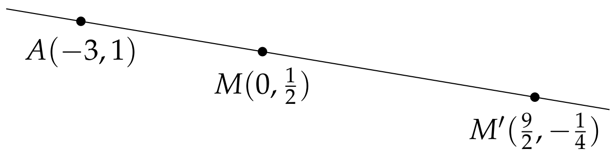
The ratio of the area of the triangles can now be computed.

```
> (/ (triangle-area A1 B1 C1)
     (triangle-area A B C))
7/3
```

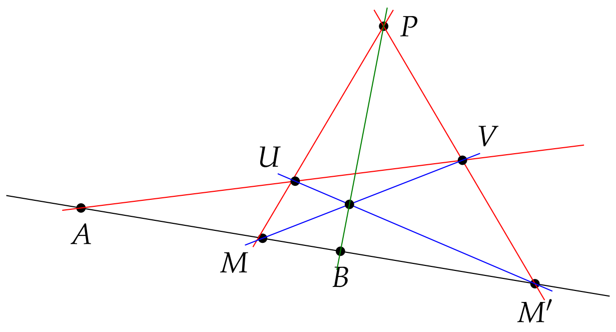
An interesting project would be to study this ratio for different triangles, and then for different values of k. What can you discover?

3.8 Constructing harmonic points

Four collinear points A, B, M, M' are said to be *harmonic* if their anharmonic ratio $(ABMM')$ is -1. Given A, M and M', where would we have to place B in order to make four harmonic points A, B, M, M'?



We can find B by choosing a center of perspective P and making the following construction.



The center of perspective P and the transversal line AV can be anywhere. Once they are decided, the rest of the construction follows. Begin by defining the given points.

```
Welcome to Racket v6.1.1.
> (define A (point -3 1))
>(define M (point 0 1/2))
> (define M1 (point 9/2 -1/4))
```

A convenient choice of P is $(2,4)$. A good way to choose the transversal line AV is to make a pencil on A and then decide on a reasonable slope.

```
> (define P (point 2 4))
> (define penA (make-pencil A))
> (define transversal (penA 1/8))
```

Determine the points U, V by using `join`.

```
> (define transversal (penA 1/8))
> (define U (join transversal (join P M)))
> (define V (join transversal (join P M1)))
> (values U V)
(point 7/13 75/52)
(point 241/73 261/146)
```

Determine the intersection point of lines UM' and MV.

```
> (define X (join (join U M1)
                  (join M V)))
> X
(point 241/168 89/84)
```

One more step to find B: join lines AM and PX.

```
> (define B (join (join A M)
                  (join P X)))
> B
(point 9/7 2/7)
```

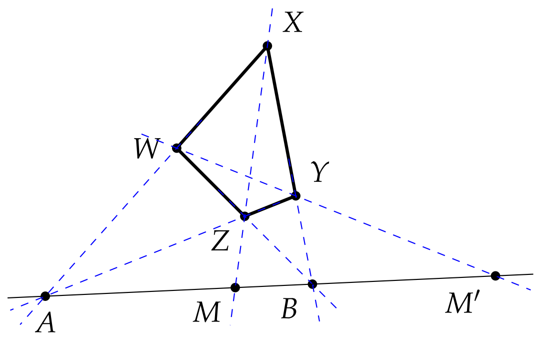
Calculate the anharmonic ratio $(ABMM')$.

```
> (anharmonic-ratio A B M M1)
-1
```

Beautiful. The points A, B, M, M' are harmonic.

3.9 Another harmonic construction

Taking the previous idea but looking at it from another point of view: any quadrilateral $WXYZ$ can be used to construct a range of harmonic points.



Joining sides XW and YZ gives point A, while joining sides WZ and XY gives point B. Finally M and M' are constructed from the diagonal lines WY and XZ. Study this for the paralellogram $W(-1, \frac{1}{2})$, $X(\frac{1}{3}, 2)$, $Y(\frac{3}{4}, -\frac{1}{5})$, $Z(0, -\frac{1}{2})$.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define W (point -1 1/2))
> (define X (point 1/3 2))
> (define Y (point 3/4 -1/5))
> (define Z (point 0 -1/2))
```

Find points A, B by joining sides.

```
> (define A (join (join X W)
                  (join Y Z)))
> (define B (join (join W Z)
                  (join X Y)))
> (values A B)
(point -85/29 -97/58)
(point 213/214 -160/107)
```

Find M and M' by joining diagonals to line AB.

```
> (define M (join (join X Z)
                  (join A B)))
> (define M1 (join (join W Y)
                   (join A B)))
> (values M M1)
(point -6/43 -133/86)
(point 468/127 -349/254)
```

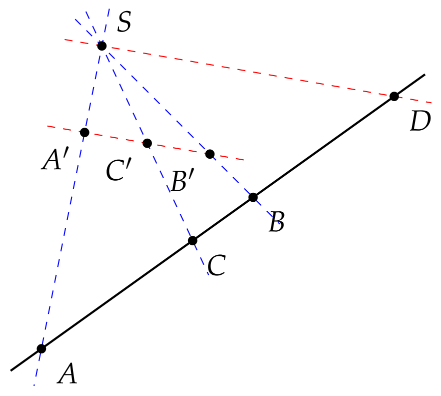
Is $(ABMM') = -1$?

```
> (anharmonic-ratio A B M M1)
-1
```

Yes it is. The points form a harmonic range.

3.10 Yet another harmonic construction

Given points $A(-3, -3)$ and $B(\frac{1}{2}, -\frac{1}{2})$, choose a point S somwhere and make the following construction.



Line SC' is a median line and lines $A'B'$ and SD are parallel. The range A, B, C, D will be harmonic. Let's try it. Define symbols for the given points and choose a good place for S.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define S (point -2 2))
> (define A (point -3 -3))
> (define B (point 1/2 -1/2))
```

Use `make-range` to choose A' along SA and B' along SB.

```
> (define A1 ((make-range A S) 5/2))
> (define B1 ((make-range B S) 2/5))
> (values A1 B1)
(point -16/7 4/7)
(point -3/14 3/14)
```

Now calculate the median line SC' and the point C.

```
> (define C1 ((make-range A1 B1) 1))
> (define C (join (join A B)
                  (join S C1)))
> C
(point -1/2 -17/14)
```

We need one more point: D. Construct a line on S that is parallel to $A'B'$. This is easily done with `make-pencil`.

```
> (define penS (make-pencil S))
> (define s (slope (join A1 B1)))
> (define D (join (join A B)
                  (penS s)))
> D
(point 17/6 7/6)
```

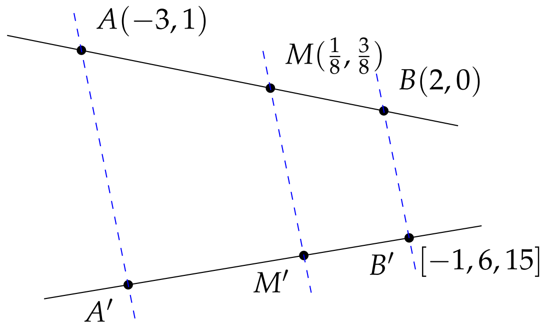
Verify that the points A, B, C, D are harmonic.

```
> (anharmonic-ratio A B C D)
-1
```

That's it!

3.11 Division ratio and parallel projection

When points are projected by means of parallel lines from one line to another, the lengths of segments defined by these points does not, in general, remain the same. However, the division ratio k of three points *does* remain the same.



Here the lines AA', BB' and MM' are parallel. We want to show that M divides AB with the same proportions as the way M' divides $A'B'$, i.e.,

$$AM : MB = AM' : M'B'$$

or $k = k'$. Begin by defining symbols for the given points and the given line $[-1, 6, 15]$.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define A (point -3 1))
> (define B (point 2 0))
> (define M (point 1/8 3/8))
> (define m (line -1 6 15))
```

To construct the necessary parallel lines, we make pencils on the given points A, M, B and we make sure to choose lines from these pencils having the same slope. Of course the slope is arbitrary, so we just choose a convenient value of -5.

```
> (define penA (make-pencil A))
> (define penB (make-pencil B))
> (define penM (make-pencil M))
```

Use join to find the projected points.

```
> (define A1 (join m (penA -5)))
> (define B1 (join m (penB -5)))
> (define M1 (join m (penM -5)))
> (values A1 M1 B1)
(point -69/31 -89/31)
(point 21/31 -74/31)
(point 75/31 -65/31)
```

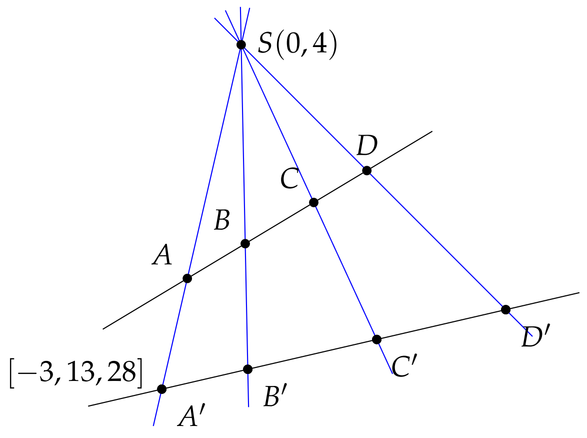
Check the division ratios of the two ranges.

```
> (division-ratio A B M)
5/3
> (division-ratio A1 B1 M1)
5/3
```

They are the same.

3.12 Anharmonic ratio and perspective

If two collinear ranges are related by perspective, then they have the same anharmonic ratio. Another way to say this is that a change of perspective does not change the anharmonic ratio. We will verify this for the following case:



We are given the center of perspective S and the line $[-3, 13, 28]$ to which the points (formidable fractions!)

$$A\left(\frac{57}{56}, -\frac{23}{56}\right), \quad B\left(\frac{19}{248}, \frac{61}{248}\right)$$

$$C\left(\frac{779}{568}, \frac{581}{568}\right), \quad D\left(\frac{19}{8}, \frac{13}{8}\right)$$

are projected by the pencil of lines emanating from S. Start by defining symbols for the given data.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define S (point 0 4))
> (define m (line -3 13 28))
> (define A (point -57/56 -23/56))
> (define B (point 19/248 61/248))
> (define C (point 779/568 581/568))
> (define D (point 19/8 13/8))
```

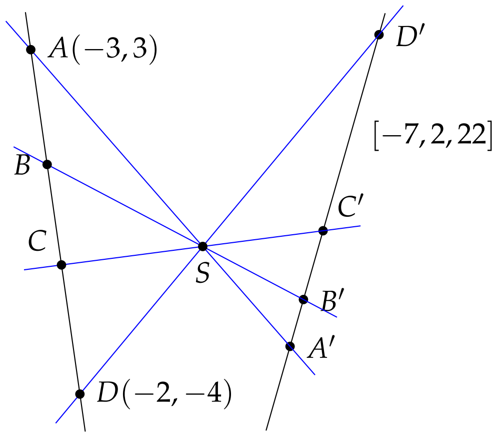
Compute A', B', C', D' by using `join`.

```
> (define A1 (join (join S A) m))
> (define B1 (join (join S B) m))
> (define C1 (join (join S C) m))
> (define D1 (join (join S D) m))
> (values A1 B1 C1 D1)
(point -3/2 -5/2)
(point 1/8 -17/8)
(point 41/16 -25/16)
(point 5 -1)
```

Find the anharmonic ratios of the two ranges.

```
> (anharmonic-ratio A B C D)
5/4
> (anharmonic-ratio A1 B1 C1 D1)
5/4
```

They are equal. Let's try this with a different perspective arrangement.



We are given $S(\frac{1}{2},-1)$, the points $A(-3,3)$, $D(-2,-4)$ and the line $[-7,2,22]$. We have to fill in the rest of the details ourselves and verify that the anharmonic ratio of A,B,C,D does not change when the points are projected over to A',B',C',D'.

First, we use `make-range` to pick out suitable points B and C.

```
$ racket -it rational-geometry.rkt
Welcome to Racket v6.1.1.
> (define S (point 1/2 -1))
> (define m (line -7 2 22))
> (define A (point -3 3))
> (define D (point -2 -4))
> (define r (make-range A D))
> (define B (r 1/2))
> (define C (r 5/3))
> (values A B C D)
(point -3 3)
(point -8/3 2/3)
(point -19/8 -11/8)
(point -2 -4)
```

Now we find the projections of the points onto line $[-7,2,22]$ using `join`.

```
> (define A1 (join (join A S) m))
> (define B1 (join (join B S) m))
> (define C1 (join (join C S) m))
> (define D1 (join (join D S) m))
> (values A1 B1 C1 D1)
(point 148/65 -197/65)
(point 130/51 -106/51)
(point 457/155 -211/310)
(point 94/23 76/23)
```

Compute the anharmonic ratios.

```
> (anharmonic-ratio A B C D)
10/7
> (anharmonic-ratio A1 B1 C1 D1)
10/7
```

They are equal!

3.13 Code listing

Many more features to can be added to this project, such as proper handling of lines with infinite slope. Support for point-conics and line-conics would open up many more problems for study. Case studies of Desargue's theorem and the extensive harmonic properties of four-point quadrilaterals would make fine projects too.

Below is the complete code listing, with unit tests. Tests are done using the `rackunit` library, which is imported into the module right at the beginning. The tests are defensive. If you modify, extend, or improve the code, it's good to have the unit tests there. They can warn you if you your changes inadvertently break something.

```
;; rational-geometry.rkt
;;
;; Rational geometry of points and lines.

#lang racket

(require rackunit)

(provide (struct-out point)
         (struct-out line)
         det
         det3
         line-equal?
         prettify
         join
         concurrent?
         collinear?
         triangle-area
         slope
         anti-slope
         make-pencil
         make-range
         division-ratio
         anharmonic-ratio)

(struct point (x y) #:transparent)
(struct line (l m n) #:transparent)

(define (det a b c d)
  (- (* a d)
     (* b c)))

(define (det3 a b c d e f g h i)
  (+ (* a (det e f h i))
     (* -1 b (det d f g i))
     (* c (det d e g h))))

;; Two lines are equal if their
;; components are proportional.
;; The following cross-products
;; must be zero:
;;
;;   [l1, m1, n1]
;;   [l2, m2, n2]
;;
;;   l1*m2 - m1*l2 = 0
;;   m1*n2 - n1*m2 = 0
;;   l1*n2 - n1*l2 = 0
;;
;; These conditions can be neatly
;; expressed using determinants.
(define (line-equal? line1 line2)
  (match-let (((line l1 m1 n1) line1)
              ((line l2 m2 n2) line2))
    (and (= 0 (det l1 m1 l2 m2))
         (= 0 (det m1 n1 m2 n2))
         (= 0 (det l1 n1 l2 n2)))))

;; Takes two lines and computes the
;; join (a point) by Cramer's method.
(define (join-lines a b)
  (match-let (((line al am an) a)
              ((line bl bm bn) b))
    (let ((D (det al am bl bm))
          (Dx (det (- an) am (- bn) bm))
          (Dy (det al (- an) bl (- bn))))
      (point (/ Dx D)
             (/ Dy D)))))

;; Compute the join of two points.
;; Divide through by n, giving
;;
;;    (l/n)*x + (m/n)*y + 1 = 0
;;
;; set p = l/n, q = m/n. we have the
;; equation x*p + y*q + 1 = 0 where
;; p, q are the unknowns. This equation
;; is satisfied at two points (x1,y1)
;; and (x2,y2). So we have two equations
;; in two unknowns:
;;
;;    x1*p + y1*q + 1 = 0
;;    x2*p + y2*q + 1 = 0
;;
;; Which can be solved by Cramer's method:
;;
;;    p = Dp/D, q = Dq/D
;;
;; However this won't work for lines
;; going through the origin, since in
;; this case n = 0. But it's clear that
;; n can be identified with D, l with Dp,
;; and m with Dq, so we don't actually
;; have to preform any division. We just
;; compute the determinants and the line
;; components follow:
;;
;;    line = [Dp, Dq, D]
;;
(define (join-points A B)
  (match-let (((point x1 y1) A)
              ((point x2 y2) B))
    (let ((Dp (det -1 y1 -1 y2))
          (Dq (det x1 -1 x2 -1))
          (D (det x1 y1 x2 y2)))
      (line Dp Dq D))))

(define (eliminate-fractions myline)
  (match-let (((line l m n) myline))
    (let ((dl (denominator l))
          (dm (denominator m))
          (dn (denominator n)))
      (let ((d (* dl dm dn)))
        (line (* d l)
              (* d m)
              (* d n))))))

(define (eliminate-factors myline)
  (match-let (((line l m n) myline))
    (let ((g (gcd l m n)))
      (line (/ l g) (/ m g) (/ n g)))))

(define (neg1 x)
  (if (< x 0) 1 0))

(define (count-negatives l m n)
  (+ (neg1 l)
     (neg1 m)
     (neg1 n)))

(define (fix-negatives myline)
  (match-let (((line l m n) myline))
    (if (< (count-negatives l m n) 2)
        myline
        (line (- l) (- m) (- n)))))

(define (prettify myline)
  (fix-negatives
   (eliminate-factors
    (eliminate-fractions myline))))

;; U, V can be two points or two lines.
;; Racket will figure out which join to
;; use.
(define (join U V)
  (cond ((and (point? U)
              (point? V))
         (prettify (join-points U V)))
        ((and (line? U)
              (line? V))
         (join-lines U V))
        (else
          (error
            'join
            "Cannot join arguments."))))

(define (triangle-area A B C)
  (match-let (((point Ax Ay) A)
              ((point Bx By) B)
              ((point Cx Cy) C))
    (* 1/2
       (+ (det Ax Ay Bx By)
          (det Bx By Cx Cy)
          (det Cx Cy Ax Ay)))))

;; Collinearity using a 3x3 determinant.
(define (collinear? A B C)
  (match-let (((point Ax Ay) A)
              ((point Bx By) B)
              ((point Cx Cy) C))
    (= 0 (det3 Ax Ay 1
               Bx By 1
               Cx Cy 1))))
```

```
;; Concurrency using 3x3 determinant.
(define (concurrent? a b c)
  (match-let (((line al am an) a)
              ((line bl bm bn) b)
              ((line cl cm cn) c))
    (= 0 (det3 al am an
               bl bm bn
               cl cm cn))))

(define (slope a)
  (let ((al (line-l a))
        (am (line-m a)))
    (- (/ al am))))

;; Returns slope of perpendicular.
(define (anti-slope a)
  (- (/ (slope a))))

;; k-pencil returns a function of slope.
;; Passing it a value for slope returns
;; line chosen from the pencil with that
;; slope. Use the equation:
;;
;;    y - y0 = k*(x - x0)
;;
;; where (x0,y0) is the pencil center.
;; Putting this into canonical line
;; form gives
;;
;;    [1, -k, k*x0 - y0]
;;
(define (make-pencil M)
  (match-let (((point Mx My) M))
    (lambda (k)
      (prettify (line (- k)
                      1
                      (- (* k Mx)
                         My))))))

;; k-range returns a function that does
;; proportional divison on the segment AB.
;; Give it a k, and it will find M.
(define (make-range A B)
  (match-let (((point Ax Ay) A)
              ((point Bx By) B))
    (lambda (k)
      (let ((a (+ 1 k)))
        (point (/ (+ Ax (* k Bx)) a)
               (/ (+ Ay (* k By)) a))))))

;; Given A, B and M, by what ratio k does
;; M divide segment AB? This is especially
;; useful for determining where M lies on
;; the line AB with respect to the points
;; A, B.
;;
;;    k = (My - Ay)/(By - My)
;;      = (Mx - Ax)/(Bx - Mx).
;;
(define (division-ratio A B M)
  (if (collinear? A B M)
      (let ((Mx (point-x M))
            (Ax (point-x A))
            (Bx (point-x B)))
        (/ (- Mx Ax) (- Bx Mx)))
      (error 'division-ratio
             "A B M must be collinear")))

;; This way of computing anharmonic ratio
;; builds on what we have above. Let k, k1
;; be the ratios
;;
;;   k = AM/BM,  k1 = AM1/BM1
;;
;; for points A, B, M and A, B, M1. Then
;; the anharmonic ratio of A, B, M, M1 is
;; the ratio of these ratios: k/k1.
(define (anharmonic-ratio A B M M1)
  (/ (division-ratio A B M)
     (division-ratio A B M1)))


;;;
;;;
;;; Tests.
;;;
;;;

;; Test determinant.
(check-equal?
  (det3 1 2 3 4 5 6 7 8 9) 0)

;; Test equality of lines.
(check-true (line-equal? (line 1 2 3)
                         (line -2 -4 -6)))
(check-true (line-equal? (line -1 1 0)
                         (line 1 -1 0)))
(check-true (line-equal? (line 1/3 2/3 7/3)
                         (line 1 2 7)))
(check-true (line-equal? (line 2 2 0)
                         (line 1 1 0)))
(check-true (line-equal? (line 1 0 0)
                         (line 5 0 0)))
(check-true (line-equal? (line 0 -2 0)
                         (line 0 1 0)))

;; Test line join.
(check-equal? (join-lines (line 1 2 3)
                          (line -3 2 1))
              (point -1/2 -5/4))
(check-equal? (join-lines (line 1 0 0)
                          (line 0 1 0))
              (point 0 0))

;; Test join points. Include cases
;; involving the origin.
(check-true
  (line-equal? (join-points (point 1 2)
                            (point -7 5))
               (line 3/8 1 -19/8)))
(check-true
  (line-equal? (join-points (point 1 1)
                            (point 2 2))
               (line -1 1 0)))
(check-true
  (line-equal? (join-points (point 0 0)
                            (point 1 1))
               (line -1 1 0)))

;; Test prettify.
(check-equal?
  (prettify (line 1/9 -4/18 -8/27))
  (line -3 6 8))

;; Test triangle area.
(let ((A (point 1 2))
      (B (point 4 3))
      (C (point 6 1)))
  (check-equal? (triangle-area A B C) -4)
  (check-equal? (triangle-area A C B) 4))

;; Test collinear.
(check-true (collinear? (point 3 8)
                        (point 0 35/4)
                        (point -9 11)))

;; Test concurrency.
(check-true (concurrent? (line -3/4 -1 11/4)
                         (line -7/6 -1 19/6)
                         (line -1/2 -1 5/2)))

;; Test slope, anti-slope.
(check-equal? (slope (line 1 2 3)) -1/2)
(check-equal? (anti-slope (line 1 2 3)) 2)

;; Test make-pencils. Make sure they work
;; on the origin.
(let ((penA (make-pencil (point 1 2)))
      (penB (make-pencil (point 0 0))))
  (check-true
   (line-equal? (penA -1)
                (line 1 1 -3)))
  (check-true
   (line-equal? (penB 1)
                (line -1 1 0))))

;; Test make-range.
(let ((ran (make-range (point 1 2)
                       (point -3 8))))
  (check-equal? (ran 1/2)
                (point -1/3 4)))

;; Test division-ratio.
(let ((A (point 1 3))
      (B (point -5 2))
      (M (point 19 6)))
  (check-equal? (division-ratio A B M) -3/4))

;; For testing.
(define (anharmonic-ratio-x A B C D)
  (let ((u1 (point-x A))
        (u2 (point-x B))
        (u3 (point-x C))
        (u4 (point-x D)))
    (/ (* (- u1 u3)
          (- u2 u4))
       (* (- u1 u4)
          (- u2 u3)))))

;; Test anharmonic ratio.
(let* ((A (point 1 3))
       (B (point -5 2))
       (ran (make-range A B))
       (C (ran -5))
       (D (ran -2)))
  (check-equal?
    (anharmonic-ratio A B C D) 5/2)
  (check-equal?
    (anharmonic-ratio A B C D)
    (anharmonic-ratio-x A B C D)))
```