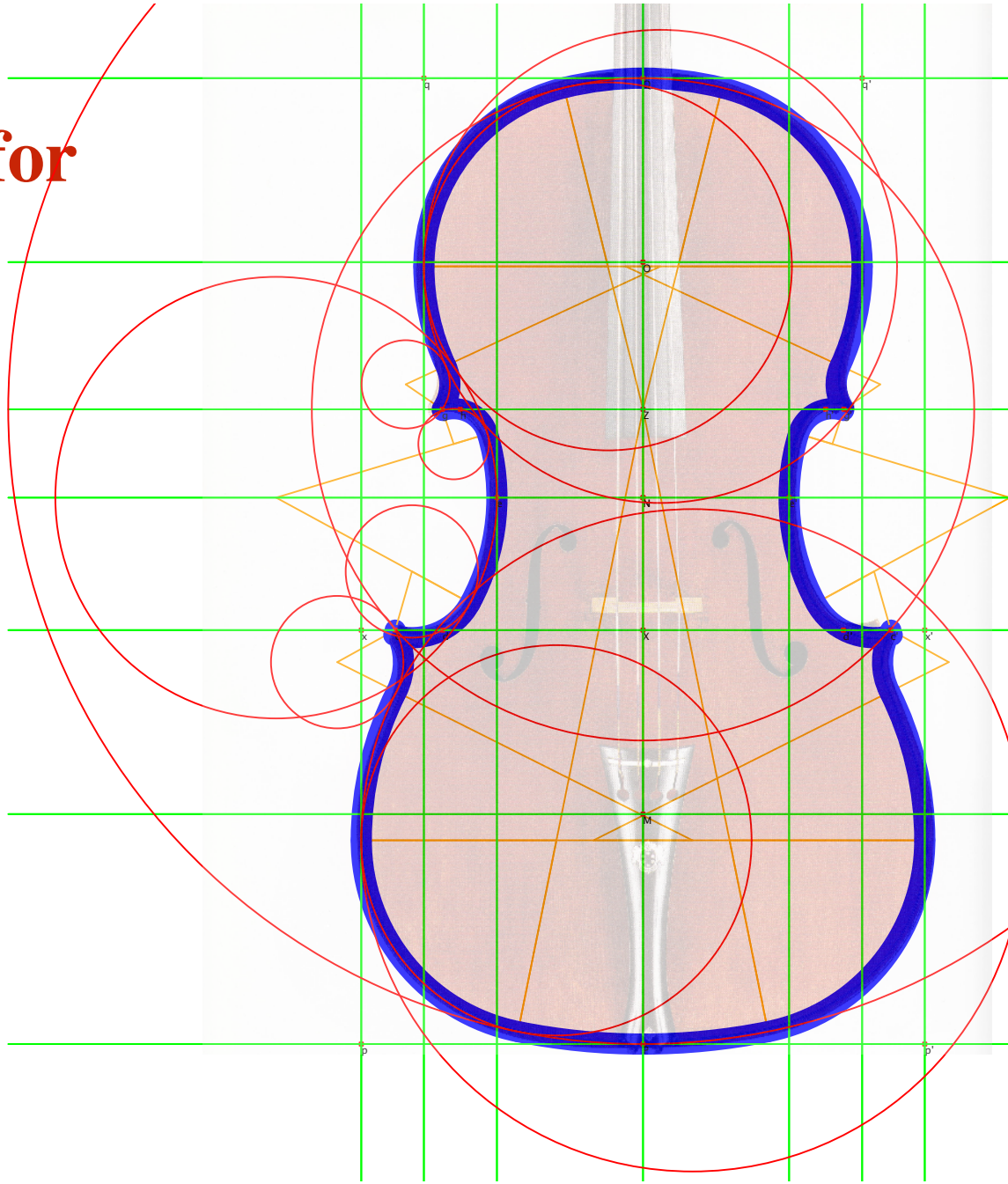
1-day course on software for "Traité de Lutherie"

January 8, 2015 (1-6pm)
January 9, 2015 (9am-2pm)

Harry Mairson
mairson@brandeis.edu



NATIONAL ENDOWMENT FOR THE HUMANITIES
NATIONAL SCIENCE FOUNDATION

Big questions:

How do you run code
to produce a drawing?

How do you write down the geometric constructions
that you have in your head? (Denis prose, turned into code)
(There's an appropriate language for everything…)

How do you structure the directions at the "top level"?
(general format)

Ground rules

This is an experiment!

So please stop me, ask questions, discussion, etc. at points where more explanation is needed…

And…

After this meeting, I'm available by email for further consultation, design debugging, explanation etc.

Also, until it drives me nuts, I can print stuff and mail it to you.

Overview

(how much of this will we get through?)

Lecture background on programming geometric descriptions.

Exercises (filling in templates, …).

Repeating designs from François Denis's "Traité de lutherie".

Cooking on your own.

What have I really implemented?

A *programmable straightedge and compass machine,* so you can code geometric designs without doing any "numerical" mathematics.

All you do (as in the seventeenth century) is Euclidean geometric constructions.

Do you need to know how to program a computer in order to use this software?

Well, yes and no.

Clearly, it helps—and I hope to give you just enough of the essential mindset, mainly to write descriptions of geometric objects.

But what you don't know or understand, you can often fake.

Example: writing HTML code.
Example: rebooting your computer.

Some (programming) language philosophy…

Language, writing, drawing are all a way of *describing*…

A *programming* language is a way of expressing algorithmic ("how to") ideas—moreover, one of *descriptions* that can be (almost as a coincidence!) be carried out by a machine.

(Freehand) drawing (also a kind of language): put on the paper what you see.
Love letters: put on the paper what you feel.
Editorial: put on the paper what you believe, with justifications.
Grant proposal: (see just above)—-with communally understood idiom.
Assembly manual: put on the paper *how to* put the thing together—-
the reader reads the manual for directions.
Program: in effect, an automated assembly manual.

Software for "Traité de Lutherie" is just a formalization of Denis's "Seven Models" (section 2.5): the technical drawing language, rendered into executable computer code.

“System design”

Code for instrument

Geometry engine
(ruler and compass machine)

Scheme/Racket language

Some language preliminaries…

This whole system is language-based: so you can see ***what you did to get what you got.***

Parentheses everywhere (which have to match!).

They identify what goes with what—for example,

in English:

((the man) bit (the dog))
[subject, verb, object]

((fruit) flies (like a banana))
((fruit flies) like (a banana))
[a famous example]

and in math:

Is $5x+3y$
$5(x+3)y$ *or* $(5x)+(3y)$*?*

```
(* 5 (+ x 3) y)
(+ (* 5 x) (* 3 y))
```

They identify *operators* (function/procedures/actions) that are applied to *arguments.* Note the use of **prefix notation**: operations *first*, followed by the arguments.

`(+ 3 5)` evaluates to *8*
`(average 12 16)` evaluates to ***14***
`(intersect big-circle little-circle)` evaluates to *the points of intersection…*

Indentation doesn't matter (except to make it look less confusing to us).
The system visually matches parentheses (so you can see where the matching ones are).
Try following my examples… ("programming without understanding")

Expressions versus commands… they're different…

Expressions *evaluate to* something. Commands *do* something. Most of what we write are expressions, which describe something (curves, instrument outline, etc.).

What's the difference between expressions and commands?

"Add $20 to your bank balance"
(A ***command*** that does something—-you expect your balance to go up.)

"Add 2 years to your age"
(An ***expression*** that, if you are 58 years old, evaluates to 60 years, but does not make you 2 years older. Your age doesn't change.)

```
(xshift p 10)
```

An expression that describes a point that is 10mm to the right of point **`p`**. But **`p`** hasn't moved a micron…

Parentheses make things happen (i.e., applying *functions* to *arguments*).

The geometric language: Points

(see "Geometry engine guide"—this is just a brief tour…)

`(point x y)`
(Evaluates to) the point at position (x,y) on the plane (in millimeters!).
In other words, ***it makes a point.***

`(xcor p)`
The x-coordinate of point **`p`**. Then **`(xcor (point x y))`** evaluates to **`x`**.

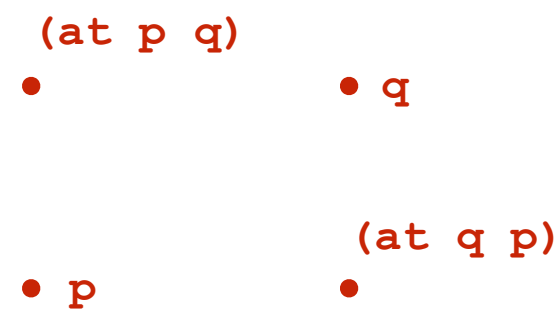
`(ycor p)`
The y-coordinate of point **`p`**.

`(at p q)`
Synonymous with **`(point (xcor p) (ycor q))`**.

`(xshift p 10)`
The point that is 10mm to the right of point **`p`**.

`(yshift p 10)`
The point that is 10mm above point **`p`**.

`(yshift p -10)`
The point that is 10mm below point **`p`**.



The geometric language: Points (2)

`(distance p q)`
The distance between points **`p`** and **`q`** (in mm).

`(midpoint p q)`
The point exactly between point **`p`** and point **`q`**.

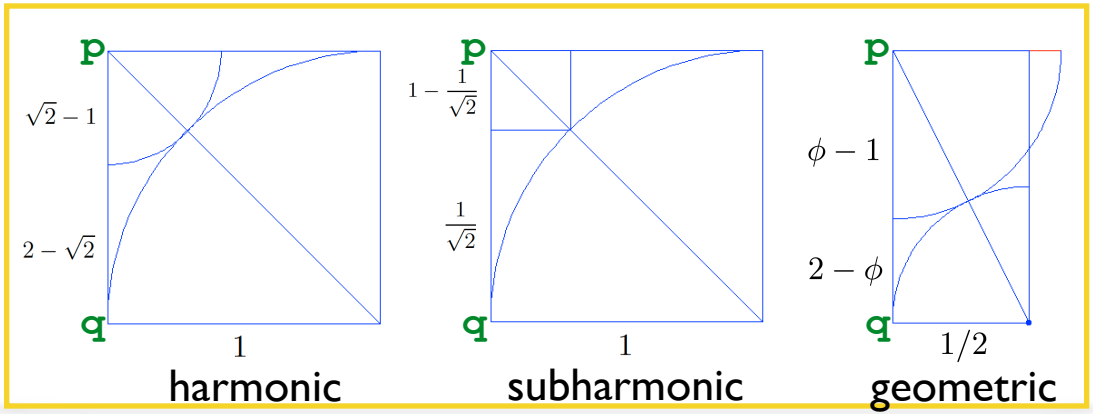
`(pointfrom p q (/ 4 5))`
The point that is 4/5th of the way from point **`p`** to point **`q`**.

`(pointfrom p q (: 2 3))`
The point that is 2 parts to 3 parts of the way from point **`p`** to point **`q`**.

`(: m n) = (/ m (+ m n))`

`(pointfrom p q geometric)`
The point that makes a geometric section between point **`p`** and point **`q`**.

`(pointfrom p q harmonic)`
`(pointfrom p q subharmonic)`
Similarly.



The geometric language: Lines

`(line p q)`
The (infinite—not a segment!) line through points **`p`** and **`q`**.

`(linefrom m p)`
The line with slope **`m`** through point **`p`**. (**`m`**=1 is 45 degree line up)

`(first-point L)`
`(second-point L)`
Recovering the two points **`p`** and **`q`**.

`(horizontal p)`
`(vertical p)`
The horizontal (vertical) lines through point **`p`**.

`(perpendicular L p)`
The line perpendicular (at right angles to) line **`L`** that is through point **`p`**.

`(bisector p q)`
The line perpendicular to **`(line p q)`** that is through point **`(midpoint p q)`**.
In fact, in the geometry engine is coded… !

```
(define (bisector p q)
  (perpendicular (line p q) (midpoint p q)))
```

The geometric language: Circles

`(circle p r)`
The circle with center (point) **`p`** and radius (number) **`r`**.

`(center C)`
`(radius C)`
The center (resp., radius) of circle **`C`**.

`(circlefrom p q)`
Synonymous with **`(circle p (distance p q))`**.

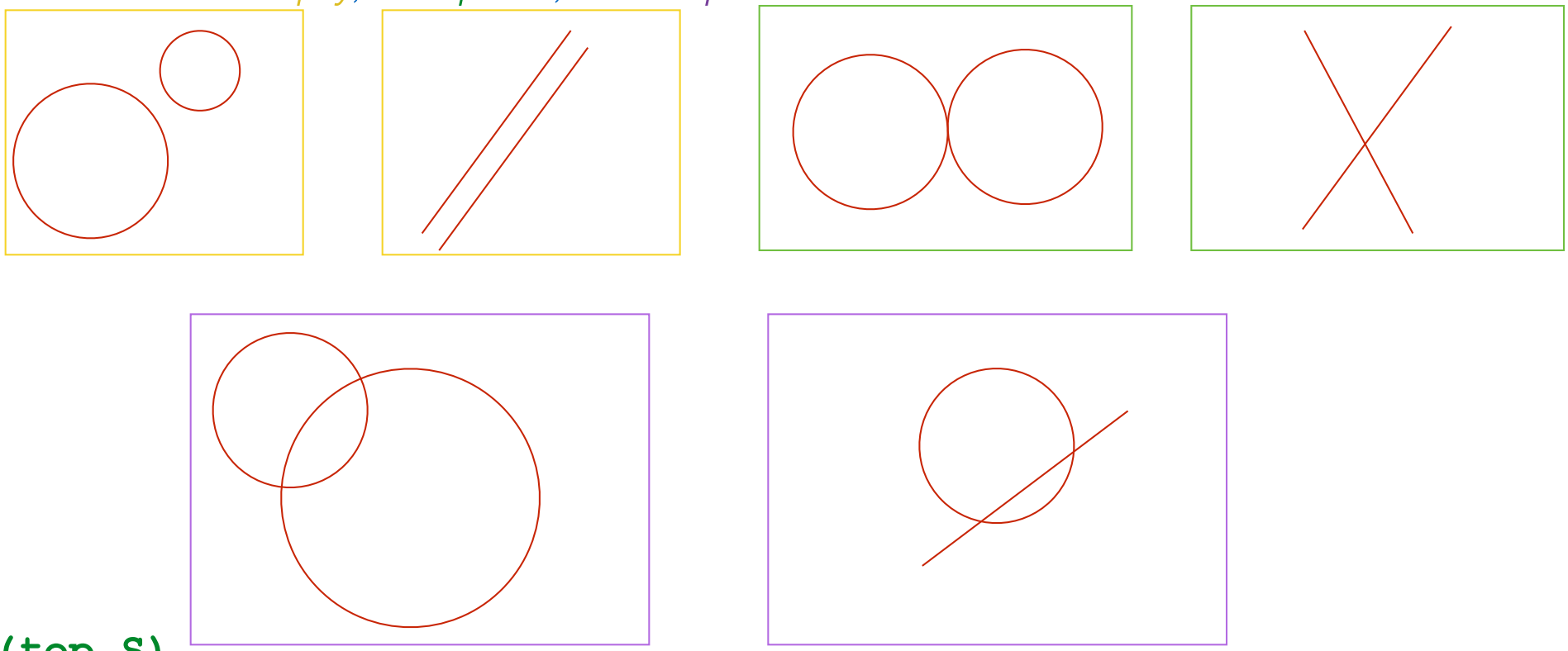
`(circlethrough p q r)`
The circle through the points **`p,q,r.`**

`(north C),(south C),(east C),(west C)`
The obvious points at the four quadrants of **`C.`**

Intersecting circles and lines

`(intersect O₁ O₂)`

Intersect two objects (either lines or circles), which then evaluates to a *list of solutions* that is either *empty*, *one point*, *or two points:*



`(top S)`
`(bottom S)`
`(left S)`
`(right S)`

Given a list **S** of solutions, the one that is the topmost/bottommost/leftmost/rightmost…
(with a compass and straightedge, you do this *automagically*…)

let*

names…

```
(let* ((v1 e1)
       (v2 e2)
       (vn en))
  b)
```

resulting object…

of geometric objects (or numbers) given by expressions…

let*

```
(let* ((v1 e1)
       (v2 e2)
        ...
       (vn en))
  b)
```

Example:

```
(let* ((x 2)
       (y 3))
  (- (* y y) (* x x)) )
```

evaluates to (3 x 3)-(2 x 2) = 5.

Geometric programming: writing down what you visualize

let*

```
(let* ((v1 e1)
       (v2 e2)
        ...
       (vn en))
  b)
```

names…

of geometric objects given by expressions…

resulting object…

Example: given points `p,q,r`, what is the circle through those points?

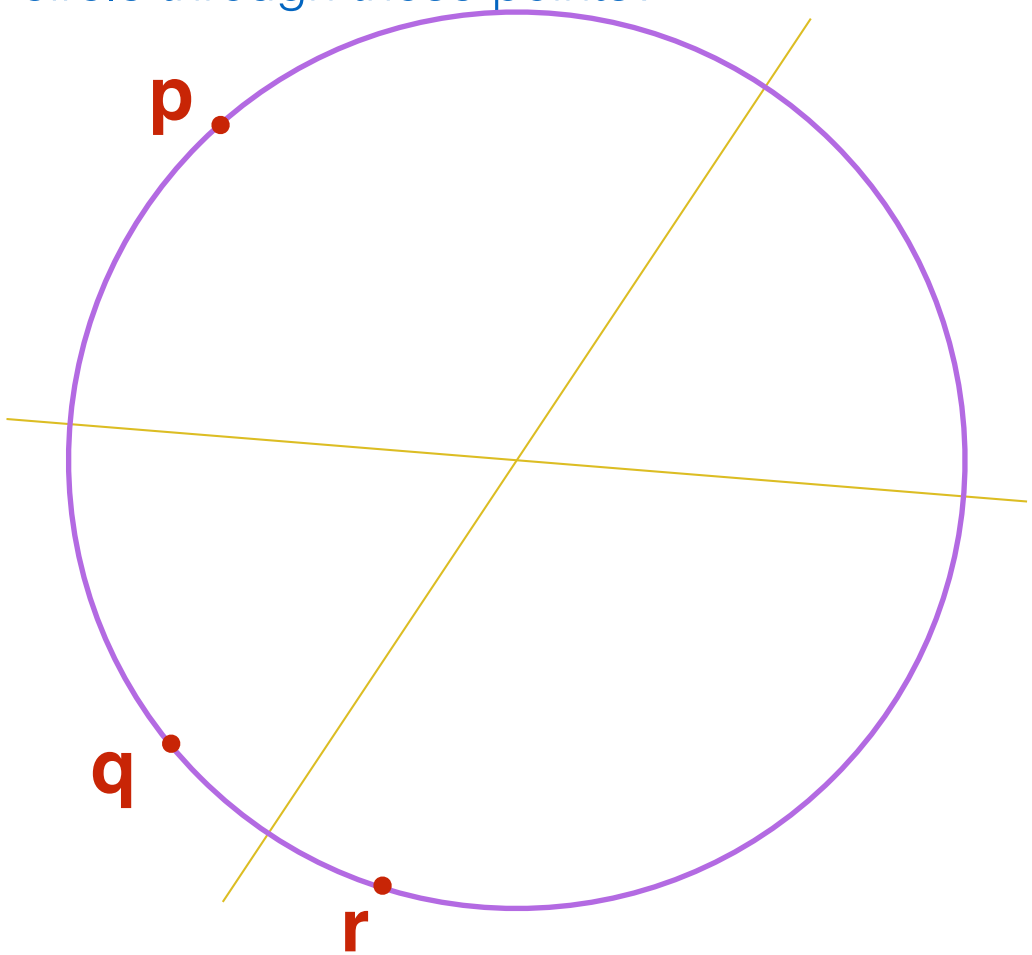
let*

```
(let* ((v1 e1)
       (v2 e2)
        ...
       (vn en))
  b)
```

Geometric programming: writing down what you visualize

(otherwise said: writing François Denis's' prose, only in code)

Example: given points `p,q,r`, what is the circle through those points?



Geometric programming: writing down what you visualize

let*

```
(let* ((v1 e1)
       (v2 e2)
        ...
       (vn en))
  b)
```

names…

of geometric objects given by expressions…

resulting object…

Example: given points `p,q,r`, what is the circle through those points?

```
(let* ((p (point x y))
       (q (point x′ y′))
       (r (point x″ y″))
       (pq (bisector p q))
       (qr (bisector q r))
       (c (intersect pq qr)))
 (circlefrom c p))
```

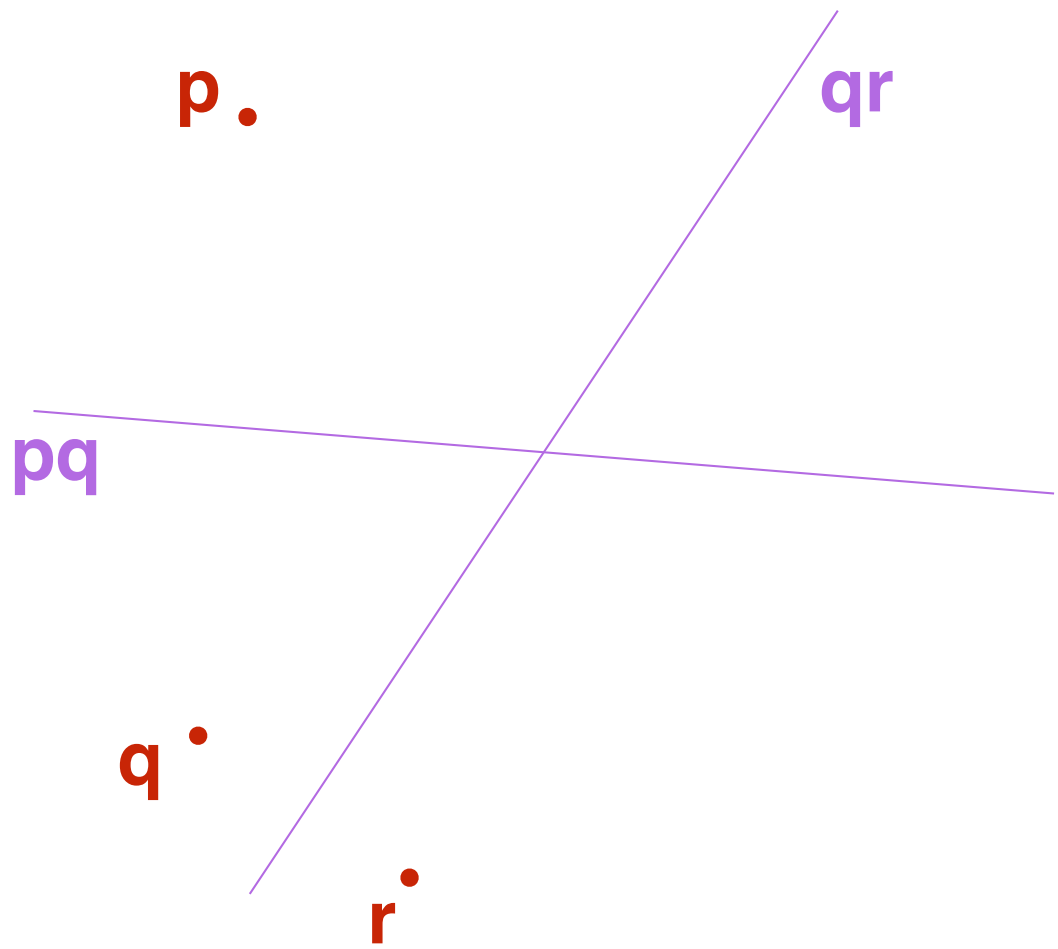
let*

```
(let* ((v1 e1)
       (v2 e2)
        ...
       (vn en))
  b)
```

Geometric programming: writing down what you visualize

Example: given points `p,q,r`, what is the circle through those points?

```
(let* ((p (point x y))
       (q (point x′ y′))
       (r (point x″ y″))
       (pq (bisector p q))
       (qr (bisector q r))
       (c (intersect pq qr)))
 (circlefrom c p))
```



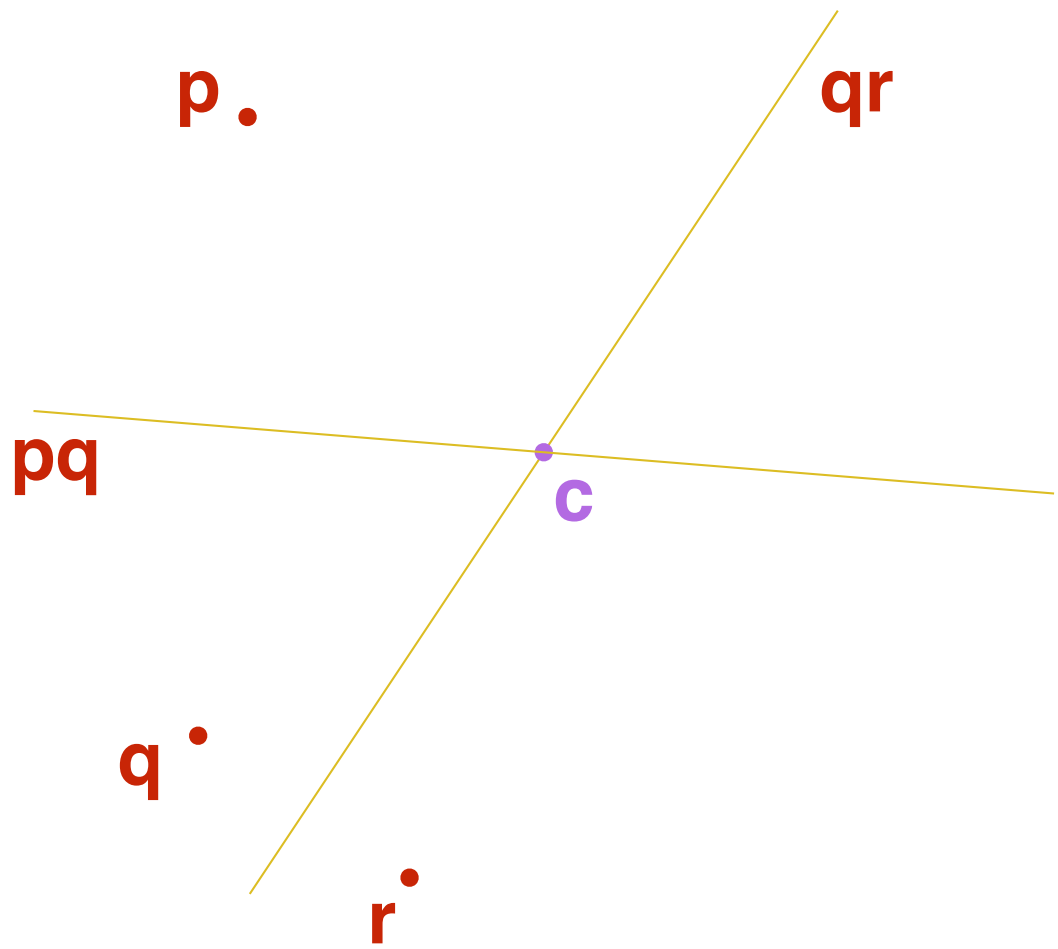
let*

```
(let* ((v1 e1)
       (v2 e2)
        ...
       (vn en))
  b)
```

Geometric programming: writing down what you visualize

Example: given points `p,q,r`, what is the circle through those points?

```
(let* ((p (point x y))
       (q (point x′ y′))
       (r (point x″ y″))
       (pq (bisector p q))
       (qr (bisector q r))
       (c (intersect pq qr)))
 (circlefrom c p))
```



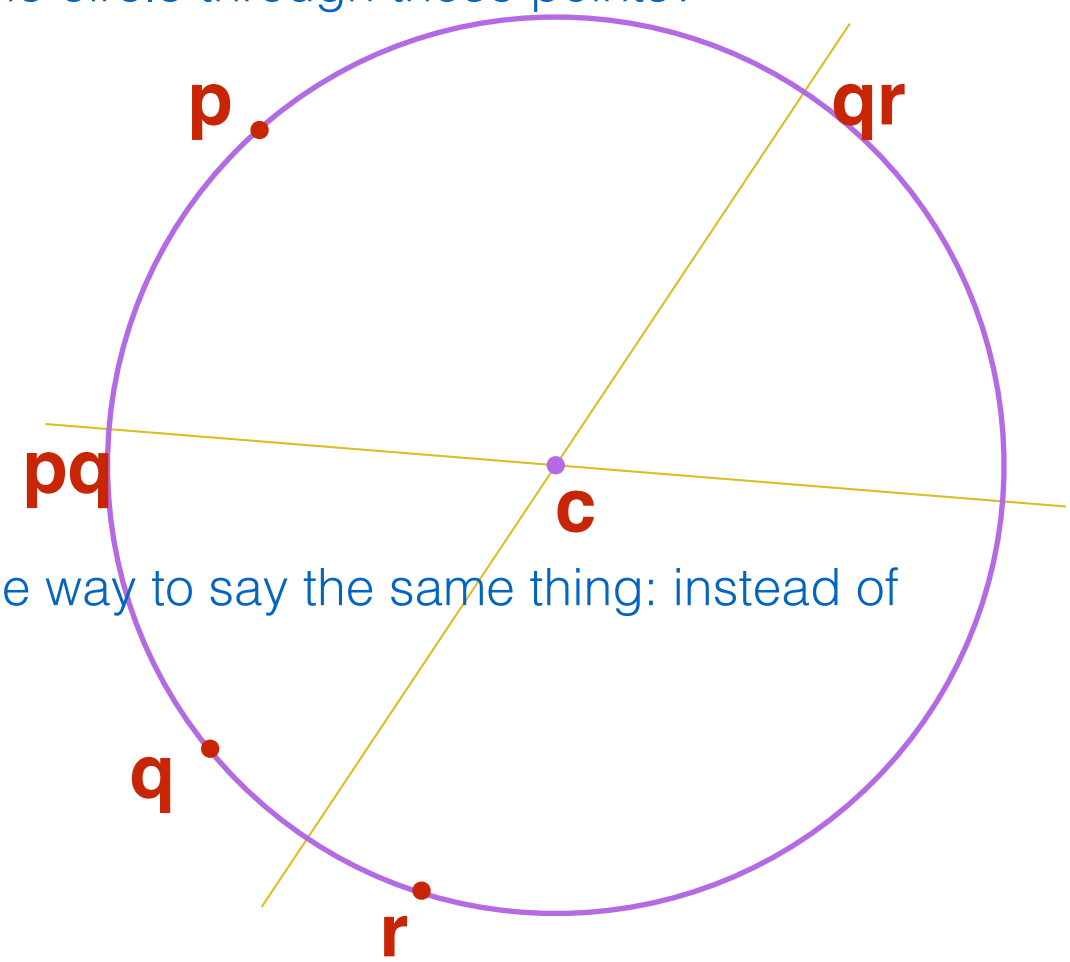
let*

```
(let* ((v1 e1)
       (v2 e2)
        …
       (vn en))
  b)
```

Geometric programming: writing down what you visualize

Example: given points `p,q,r`, what is the circle through those points?

```
(let* ((p (point x y))
       (q (point x′ y′))
       (r (point x″ y″))
       (pq (bisector p q))
       (qr (bisector q r))
       (c (intersect pq qr)))
 (circlefrom c p))
```

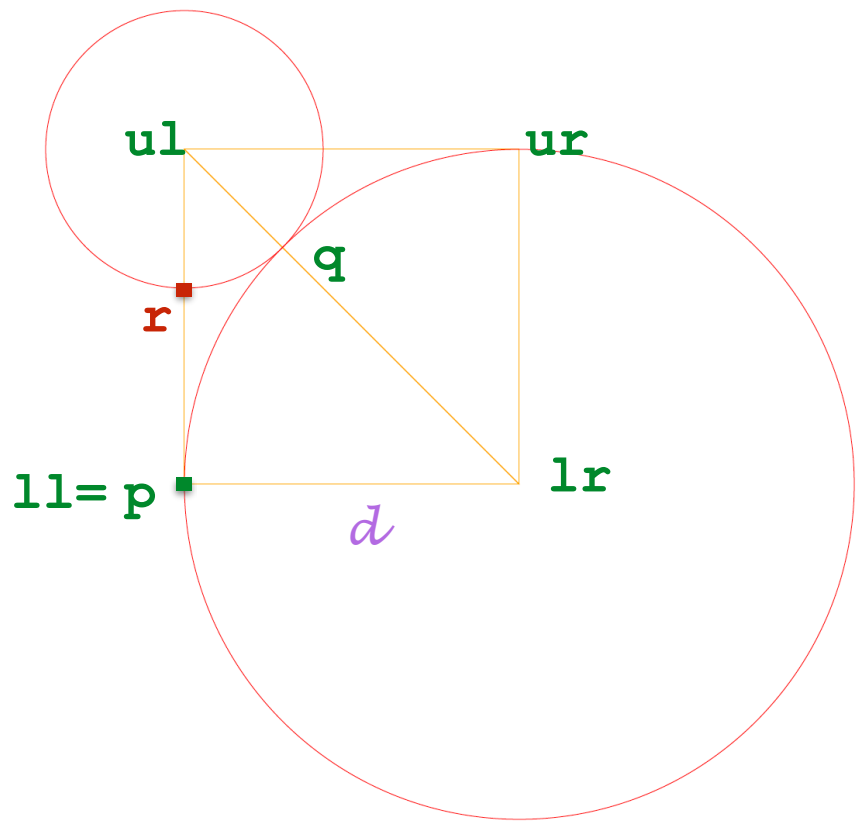


As in any language, there's more than one way to say the same thing: instead of `(circlefrom c p)` we could write…

```
(circlefrom
  (intersect (bisector p q)
             (bisector q r))
  p)
```

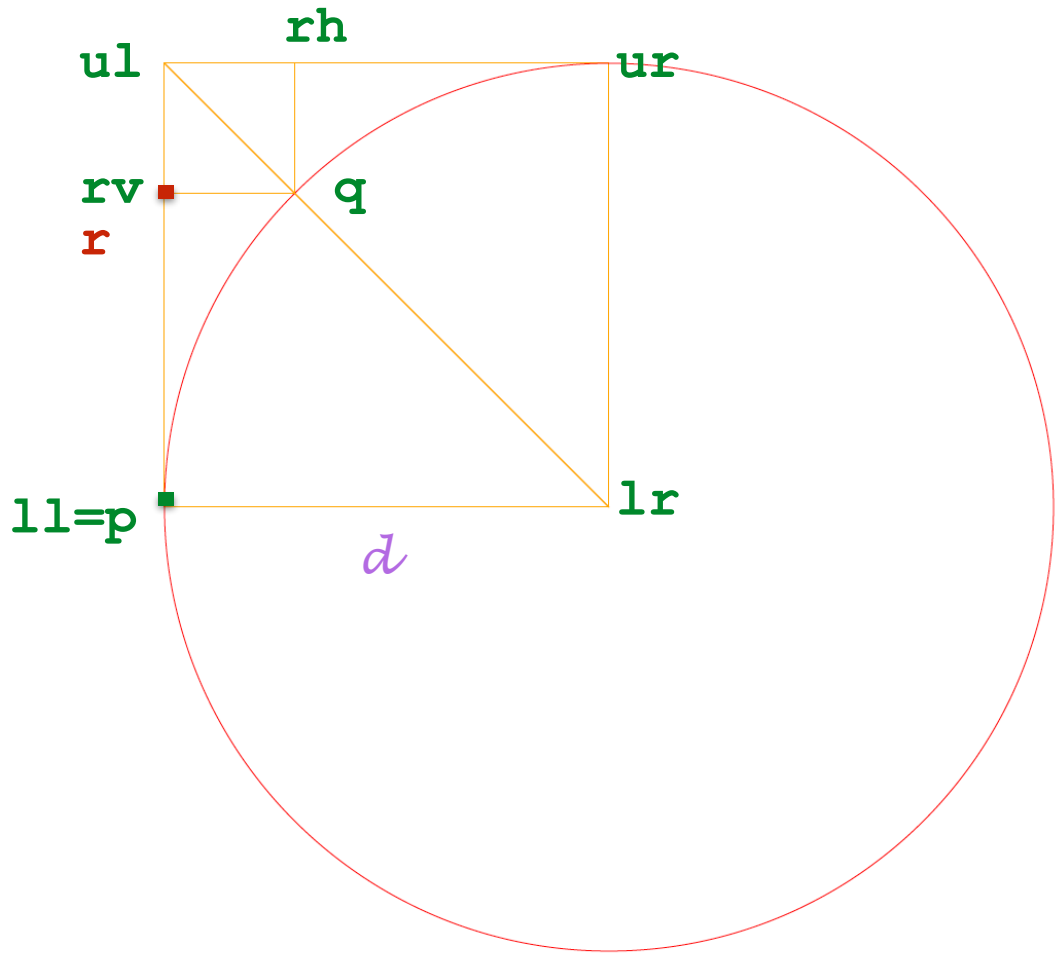
"to draw a harmonic section from p with distance d"

```
(define (harmonic p d)
  (let* ((ul (yshift p d))
         (ur (xshift ul d))
         (ll p)
         (lr (xshift p d))
         (q (top (intersect (circle lr d) (line ul lr))))
         (c (circlefrom ul q)))
    (list (polygon ul ur lr ll ul lr)
          (circle lr d)
          c
          (label "r" (south c)) )))
```



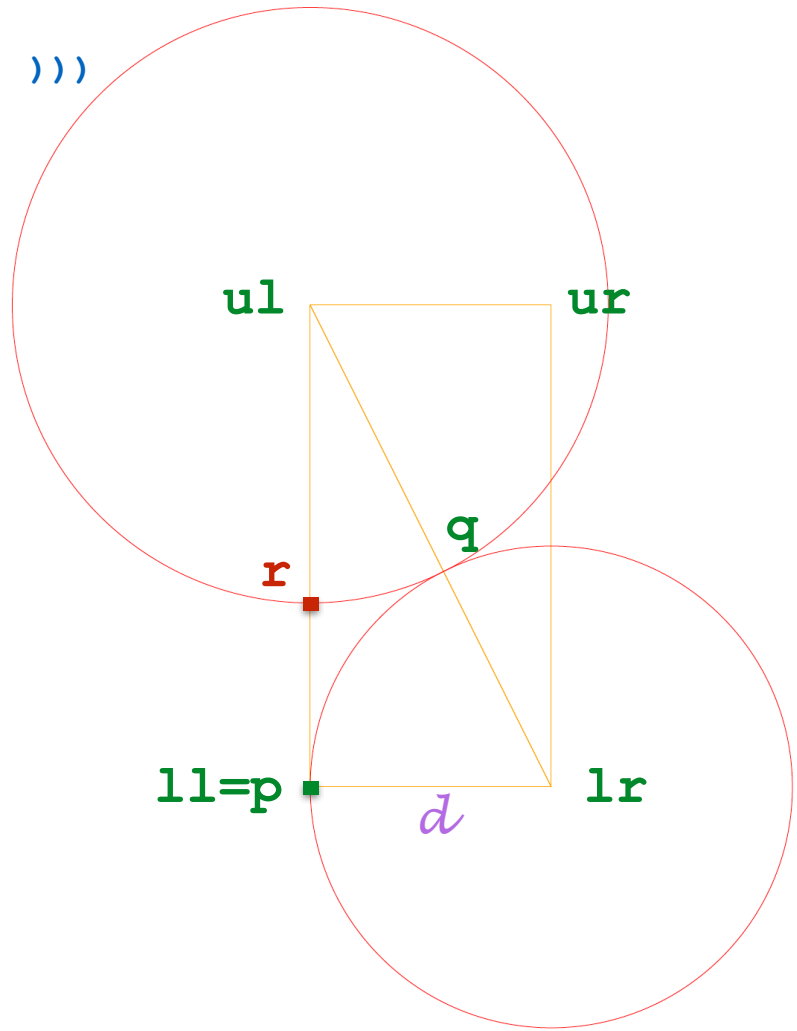
"to draw a subharmonic section from p with distance d"

```
(define (subharmonic p d)
  (let* ((ul (yshift p d))
         (ur (xshift ul d))
         (ll p)
         (lr (xshift p d))
         (q (top (intersect (circle lr d) (line ul lr))))
         (rv (intersect (horizontal q) (vertical ul)))
         (rh (intersect (vertical q) (horizontal ul))))
    (list (polygon lr ll ul ur lr ul rh q rv)
          (circle lr d)
          (label "r" rv)))
```



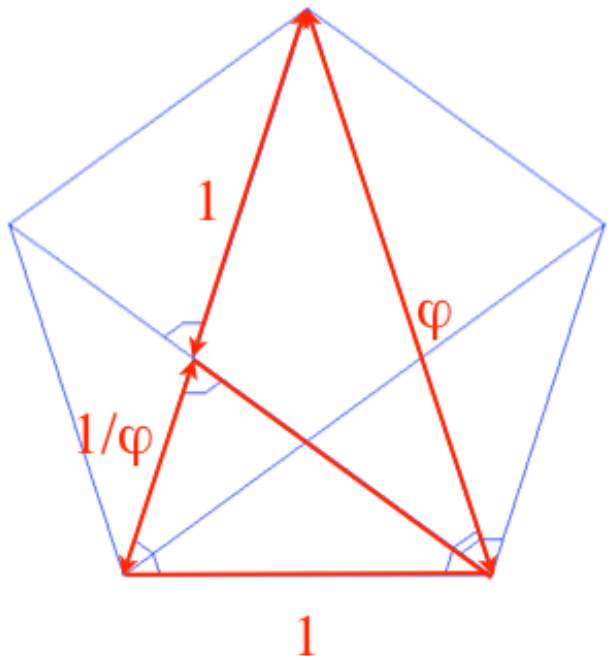
"to draw a geometric section from p with distance d"

```
(define (geometric p d)
  (let* ((ul (yshift p (* 2 d)))
         (ur (xshift ul d))
         (ll p)
         (lr (xshift p d))
         (q (top (intersect (circle lr d) (line ul lr)))))
    (list (polygon ul ur lr ll ul lr)
          (circle lr d)
          (circlefrom ul q)
          (label "r" (south (circlefrom ul q))) )))
```



Pentagons, etc. (a minor obsession of mine…)

(The big red and lower red triangles are similar.)



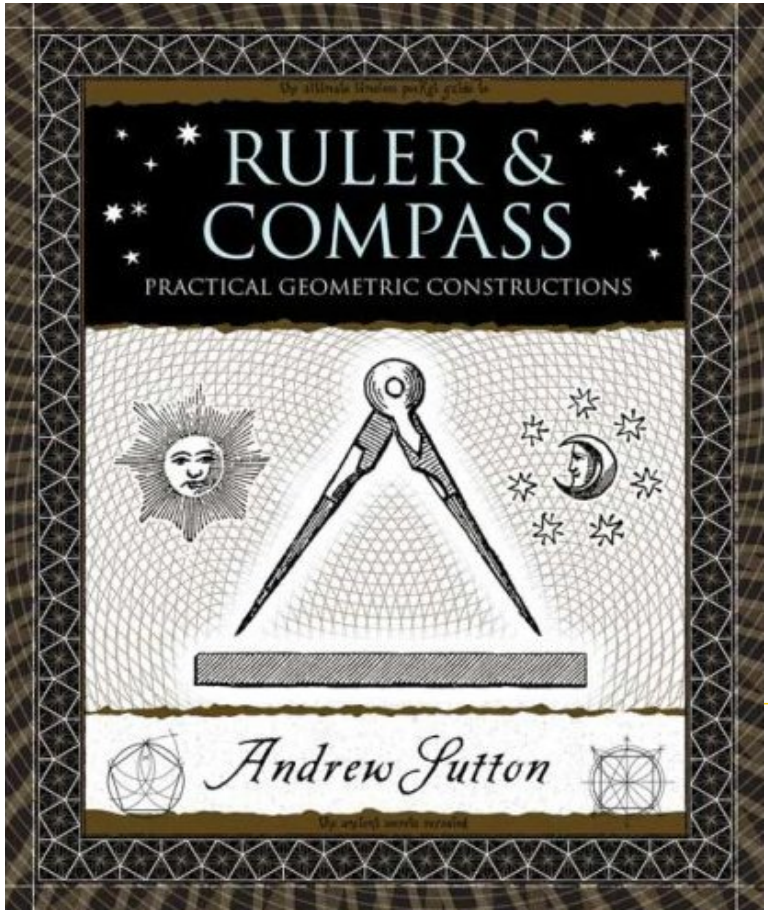
$$\varphi = 1 + 1/\varphi$$

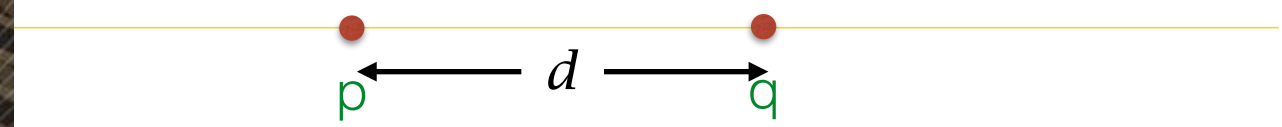
$$\varphi^2 - \varphi - 1 = 0$$

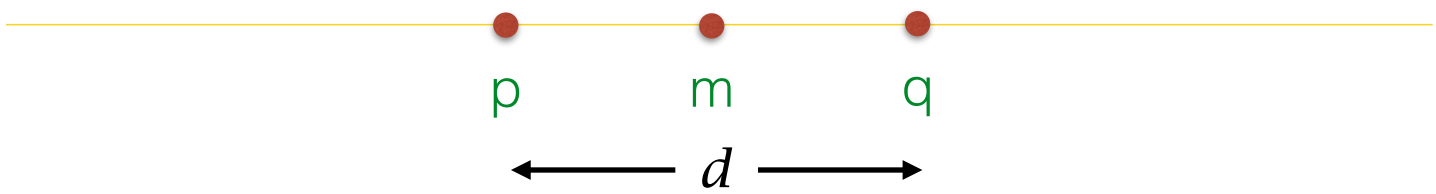
$$\varphi = \tfrac{1}{2}(1 + \sqrt{5})$$

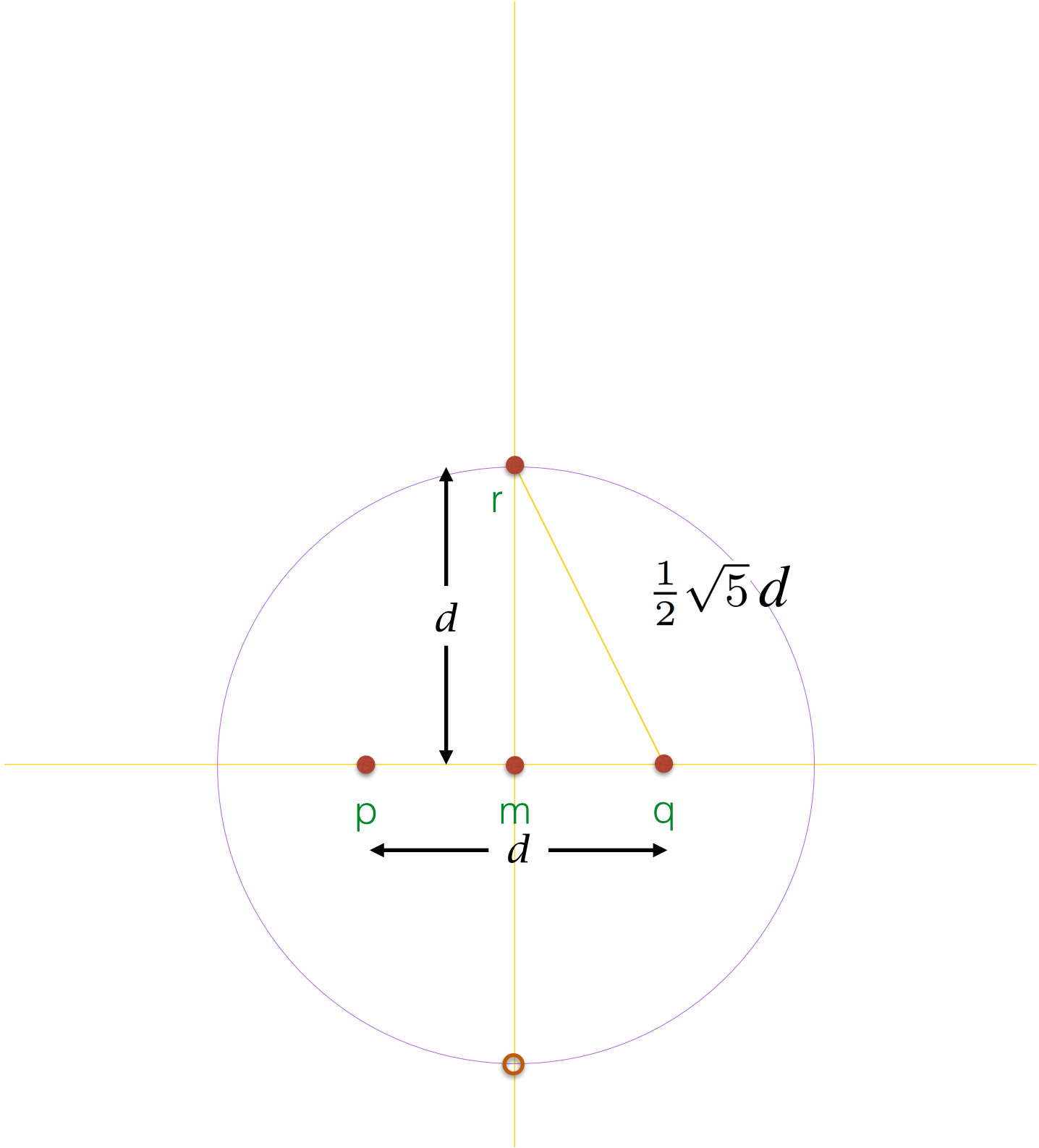
"diameter" of pentagon

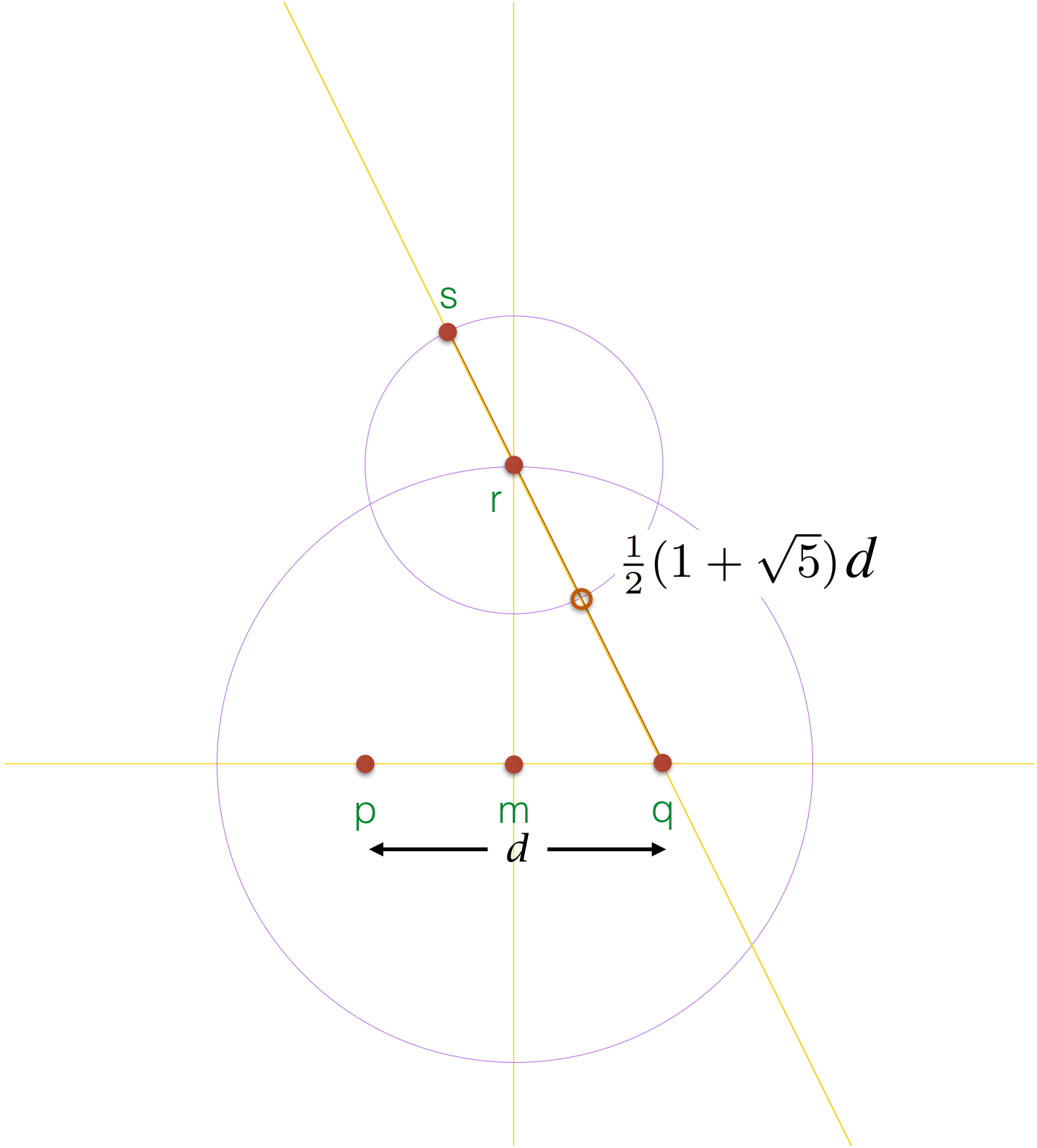
“to draw a pentagon with side d,”

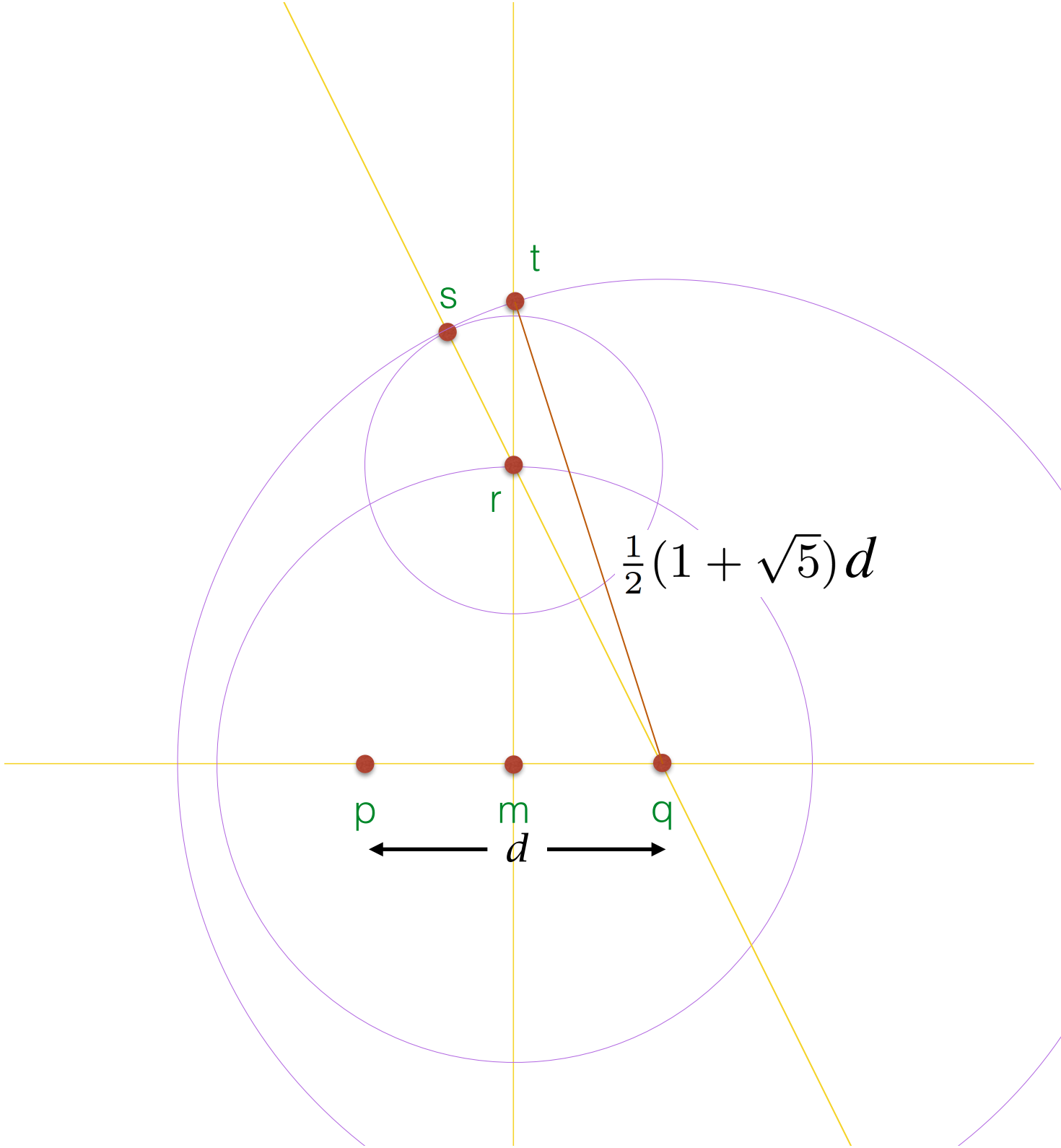


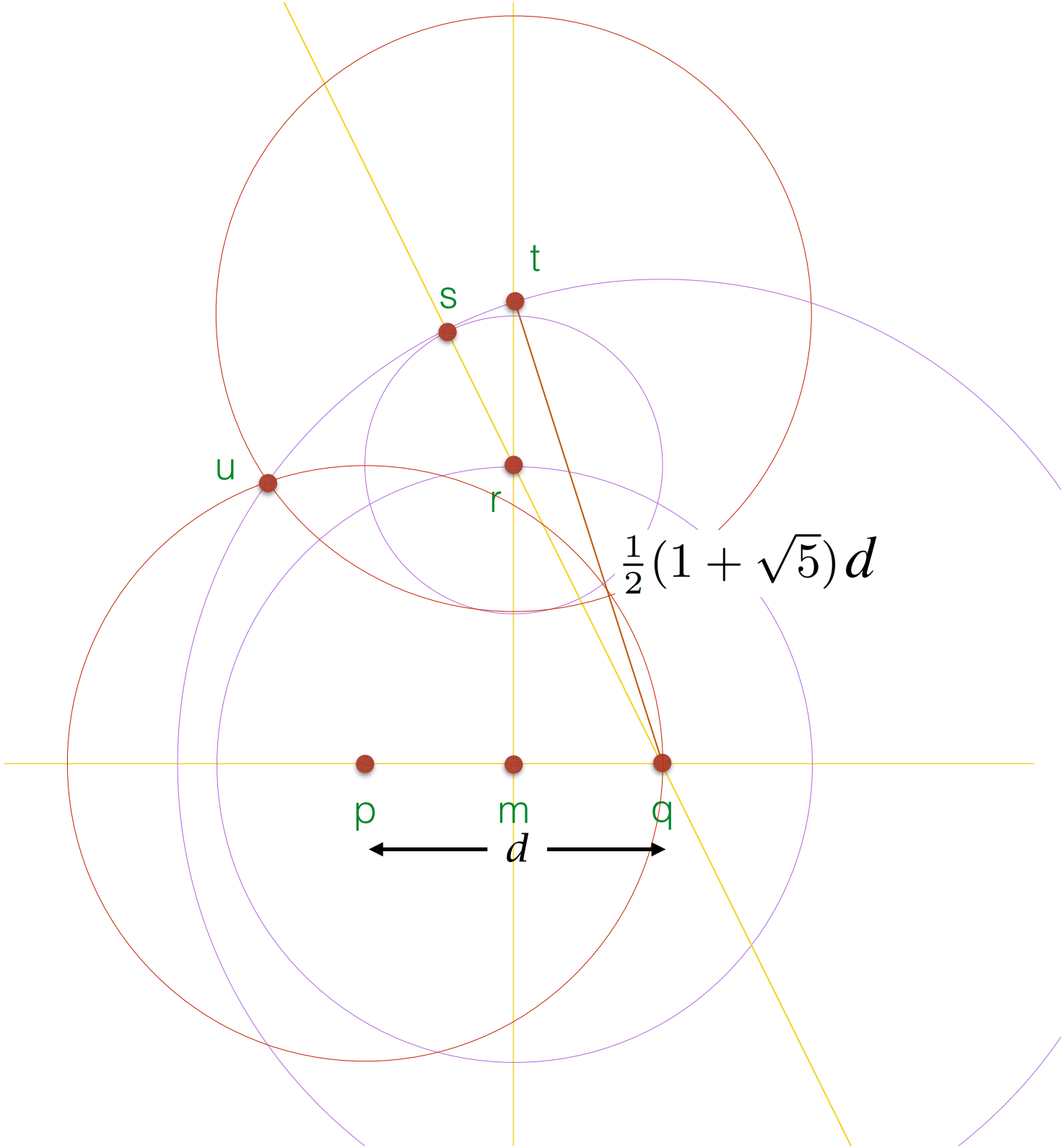


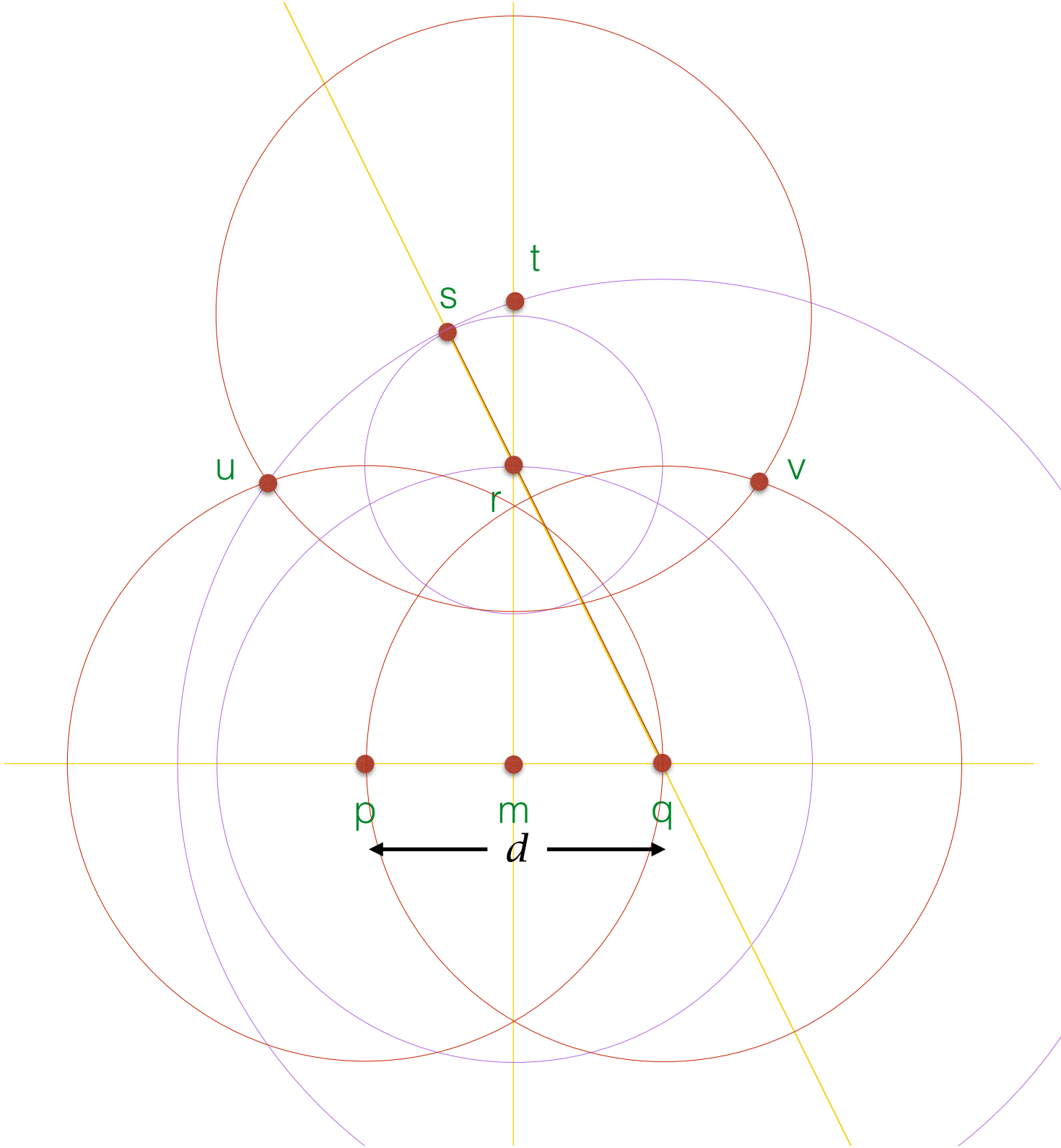

p
m
q
d


r
$\frac{1}{2}\sqrt{5}\,d$
d
p
m
q
d

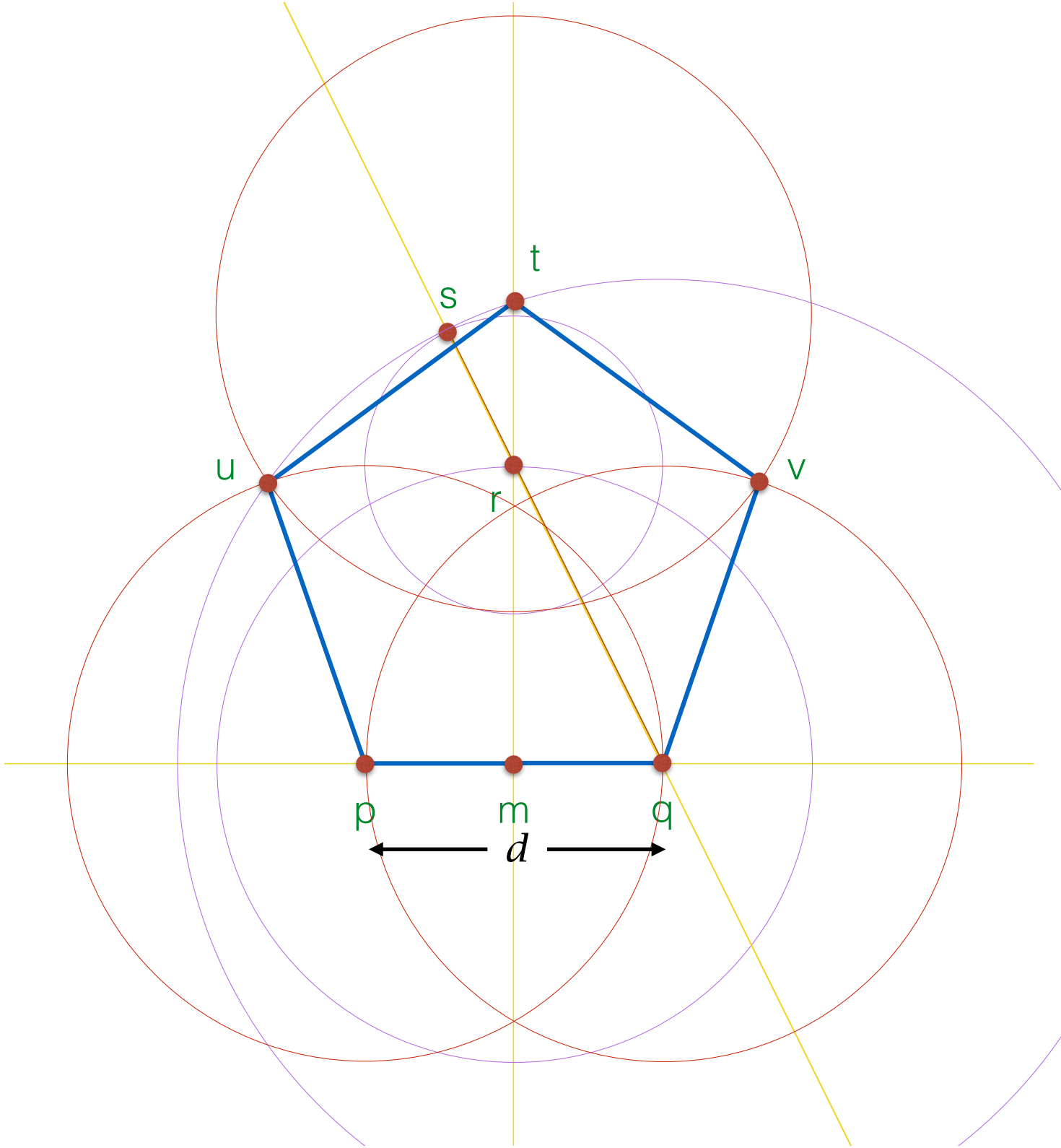

s
r
$\frac{1}{2}(1+\sqrt{5})d$
p
m
q
d


t
s
r
$\frac{1}{2}(1+\sqrt{5})d$
p
m
q
d


t
s
u
r
$\frac{1}{2}(1+\sqrt{5})d$
p
m
q
d


t
s
u
v
r
p
m
q
d

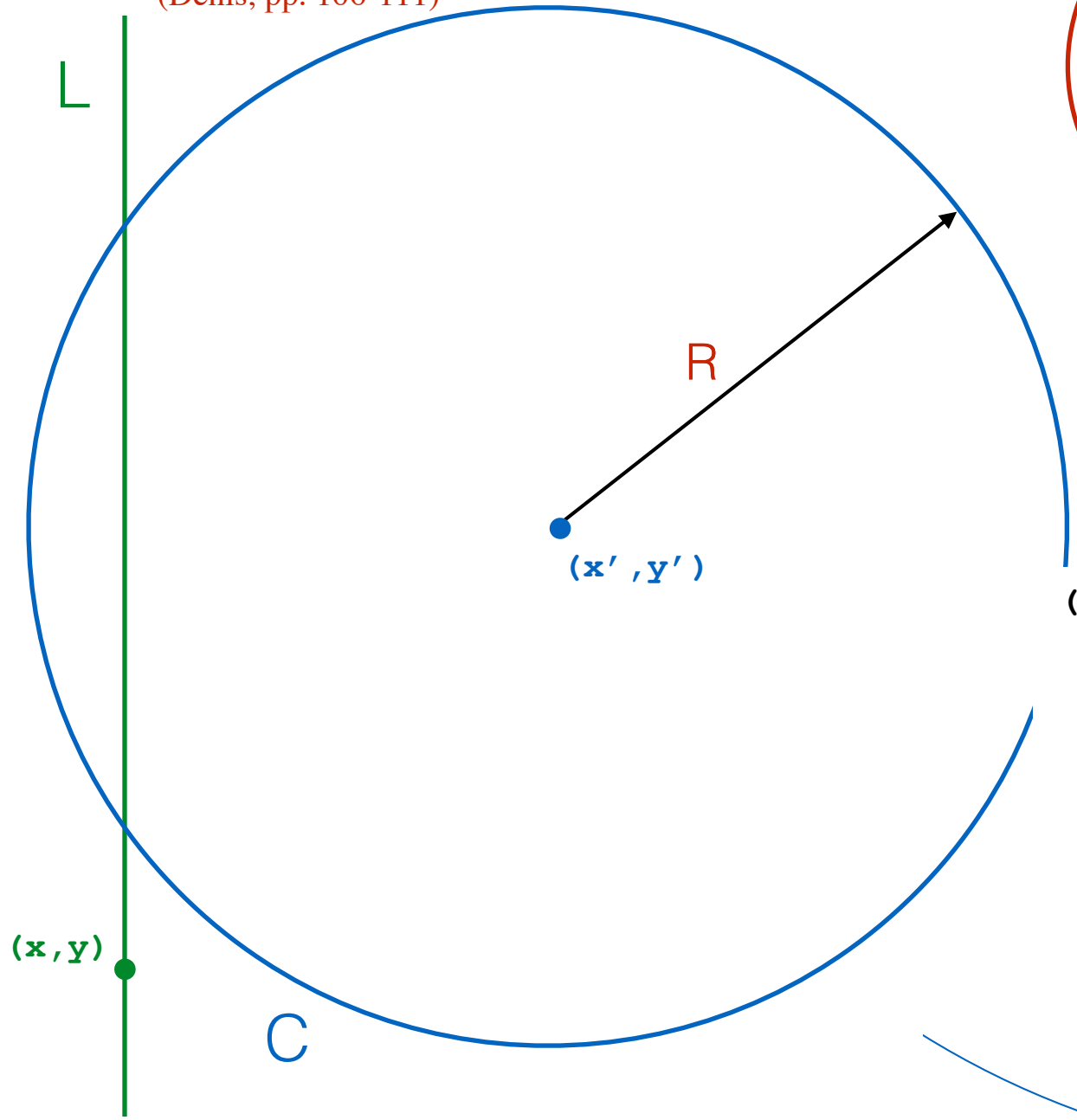
…but how do you describe all that?

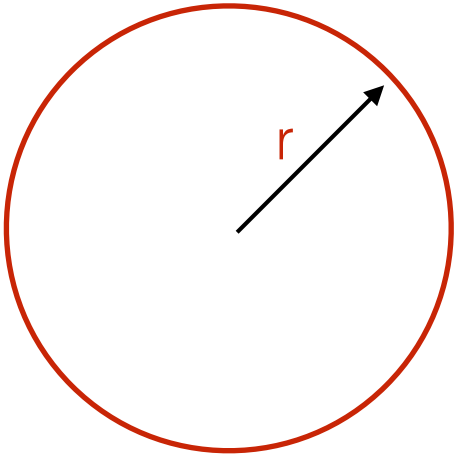


Tangency problem 1

Drawing with a compass: Three problems and their solutions

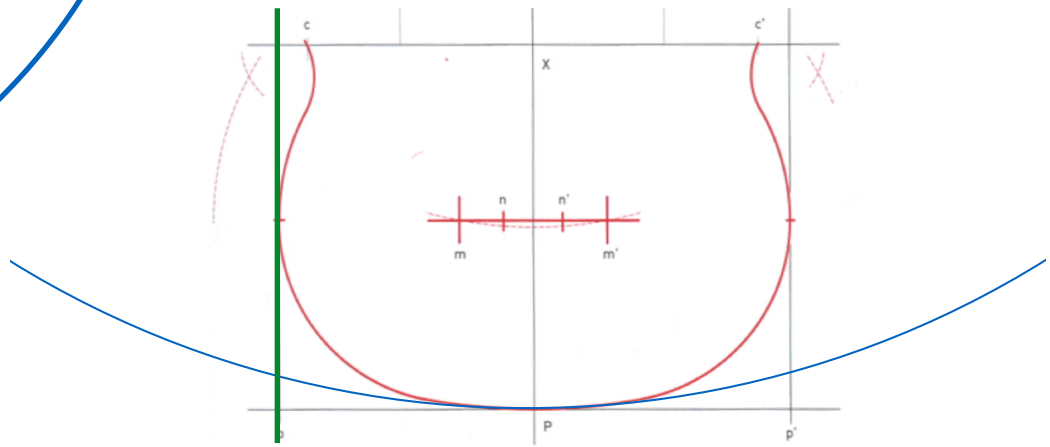
(Denis, pp. 106-111)



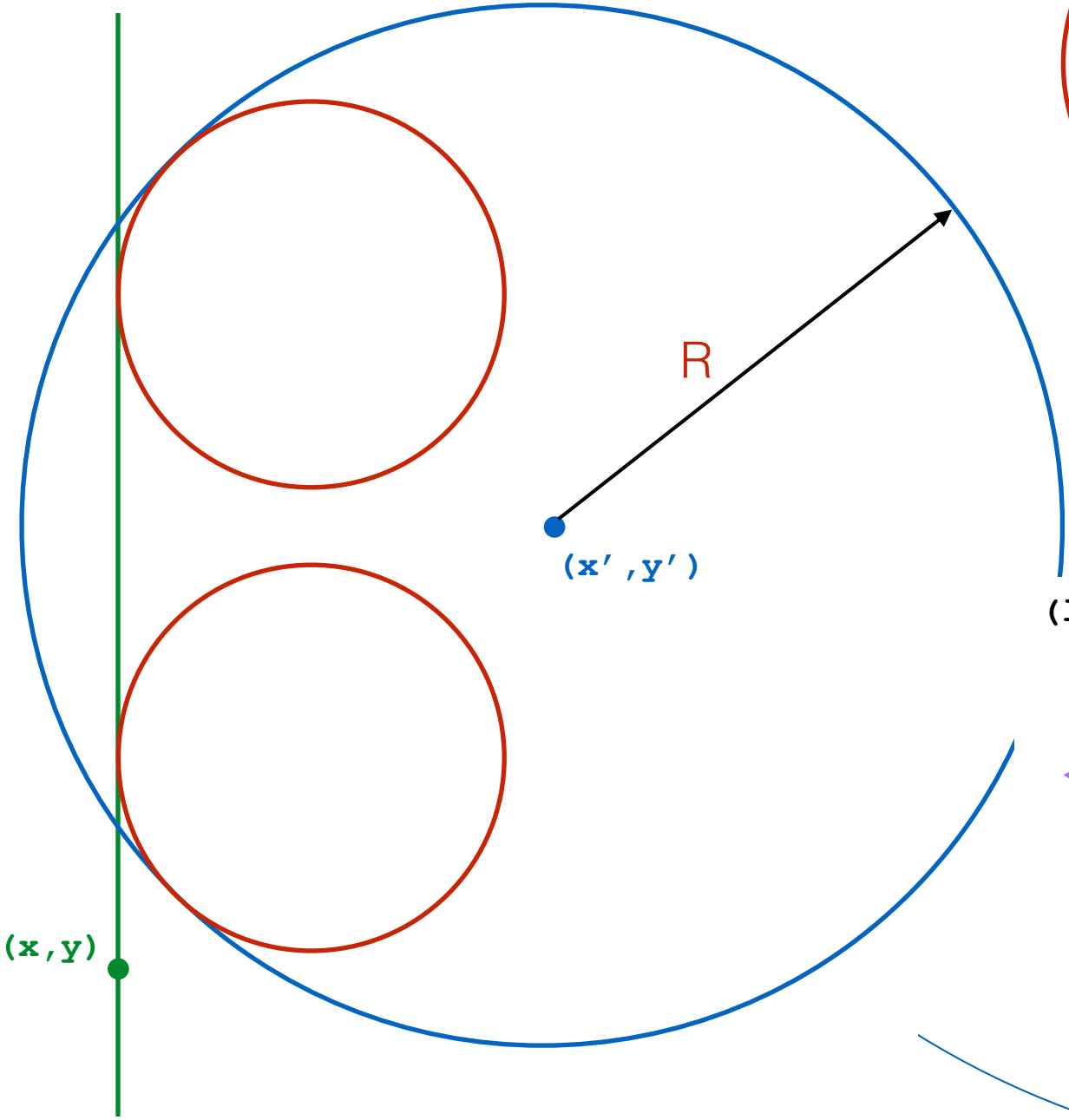


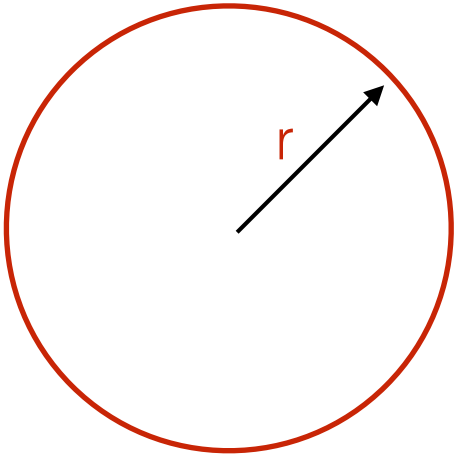
Place red circle with radius r tangent (on *right*) to green line L, *inside* blue circle C

```
(let*
  ((L (vertical (point x y)))
   (C (circle (point x′ y′) R)))

<what goes in here??>)
```



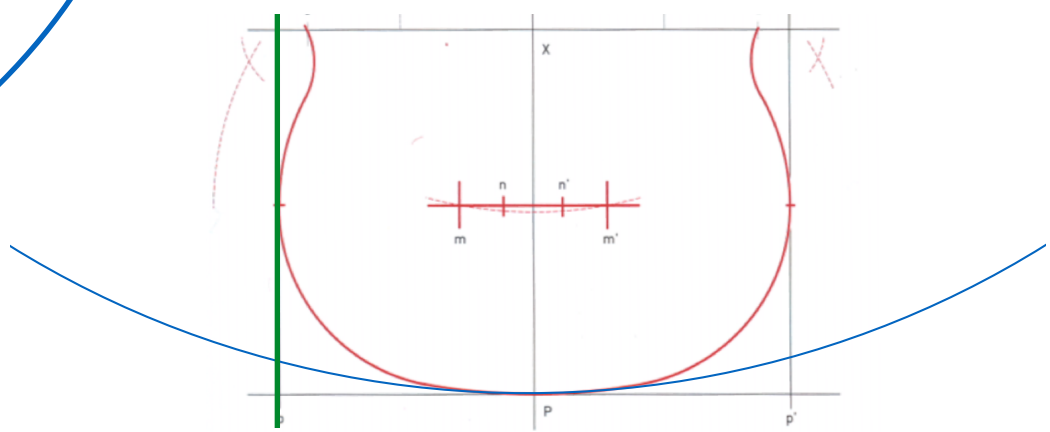
Tangency problem 1



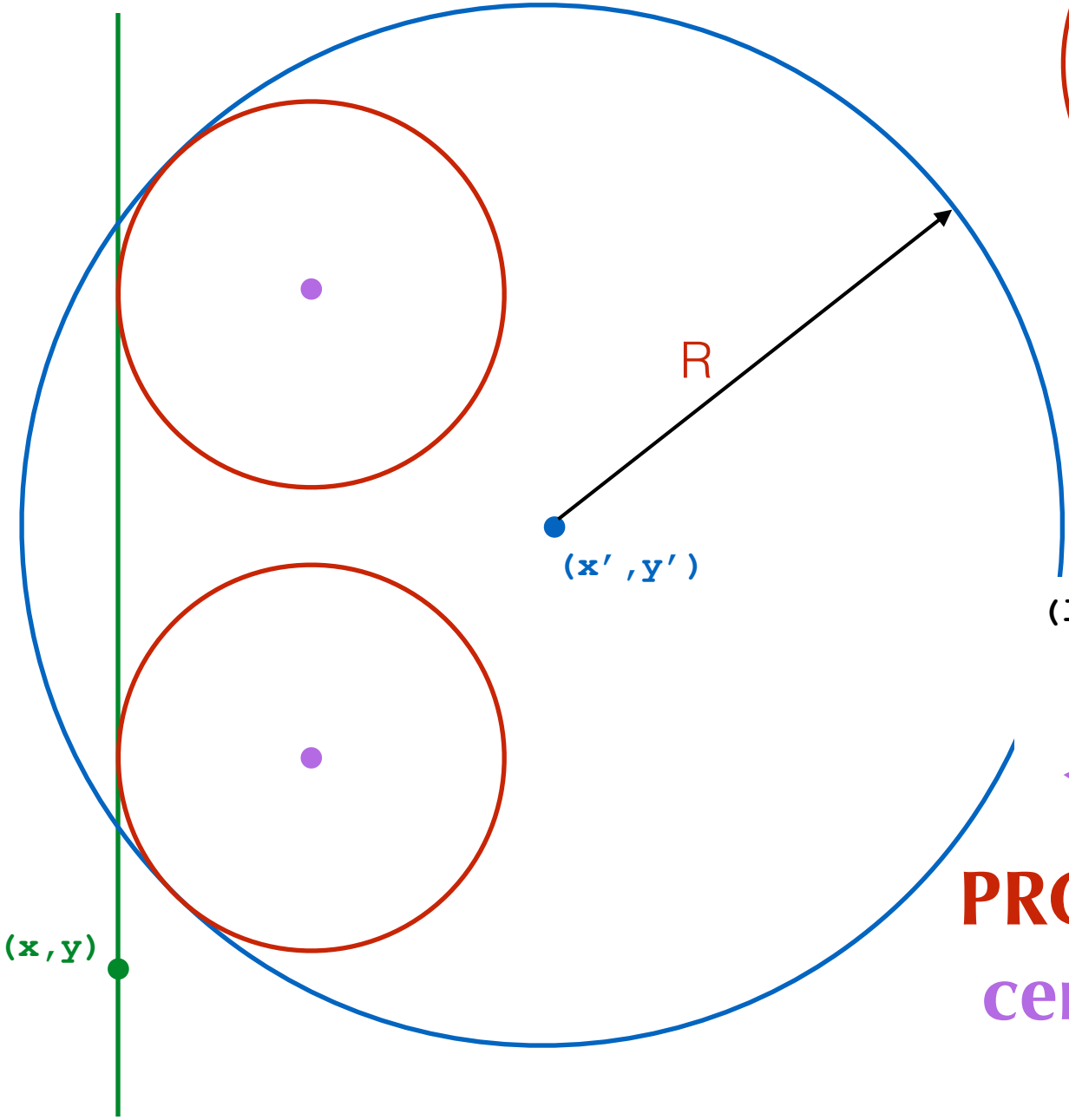


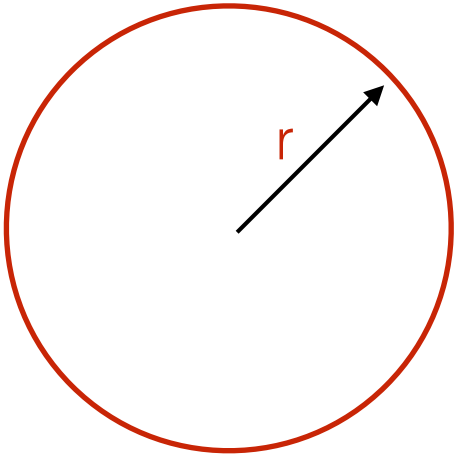
Place red circle with radius r tangent (on *right*) to green line L, *inside* blue circle C

```
(let*
  ((L (vertical (point x y)))
   (C (circle (point x′ y′) R)))

<what goes in here??>)
```



Tangency problem 1



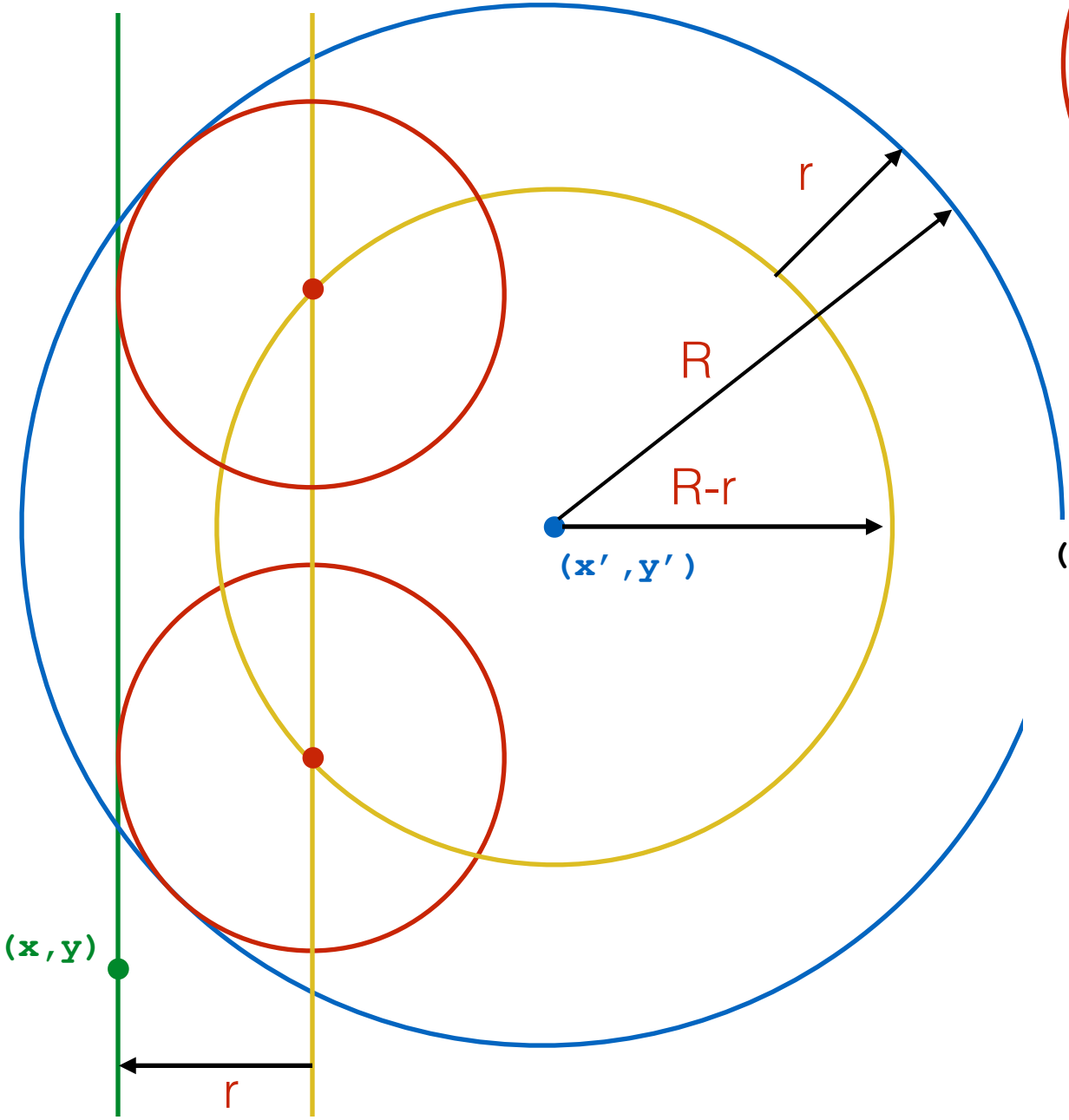


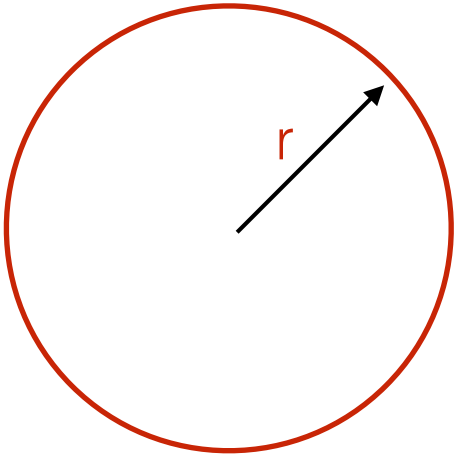
Place red circle with radius r tangent (on *right*) to green line L, *inside* blue circle C

```
(let*
  ((L (vertical (point x y)))
   (C (circle (point x′ y′) R)))

 <what goes in here??>)
```

PROBLEM: Describe the center of the red circle

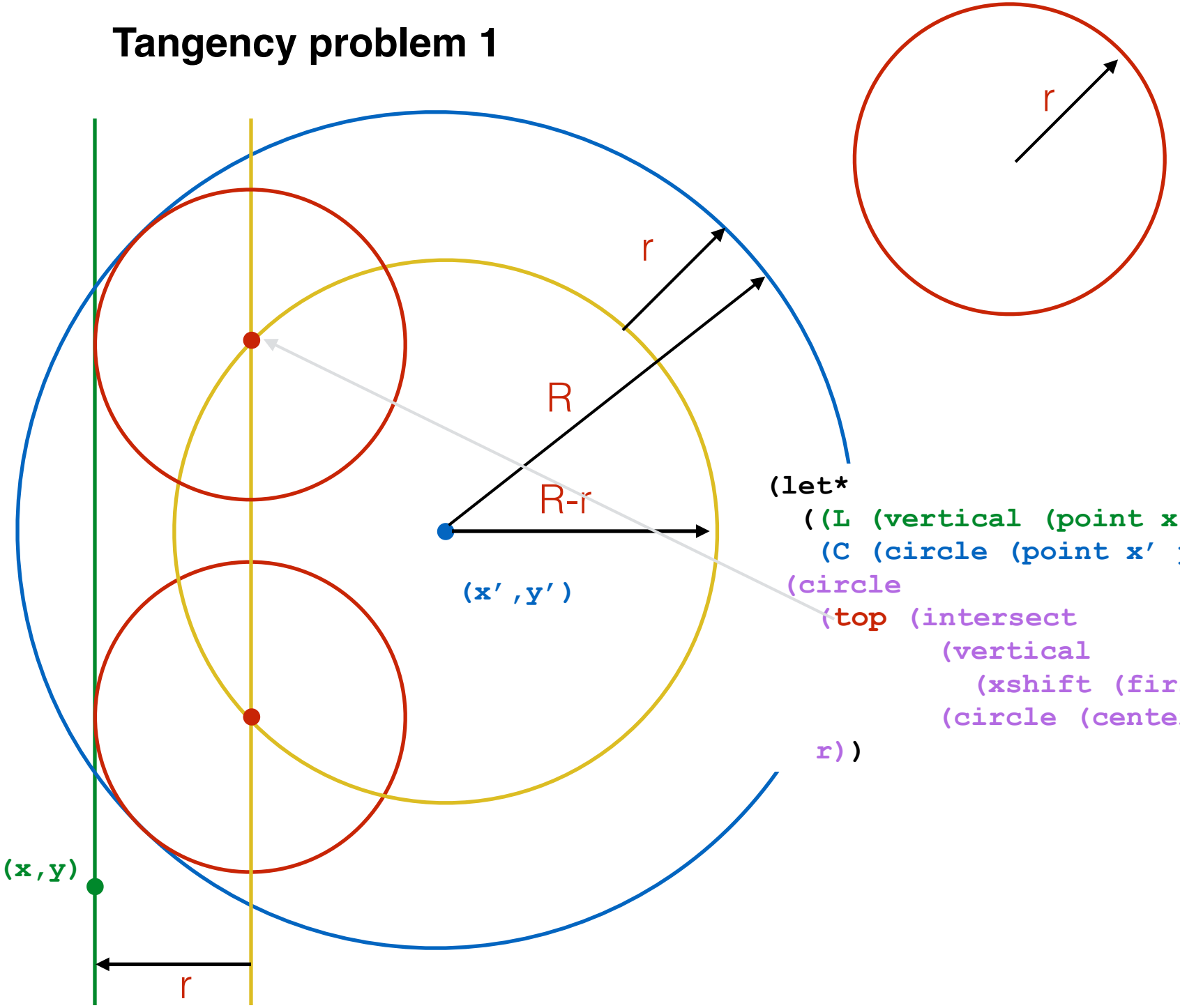
Tangency problem 1





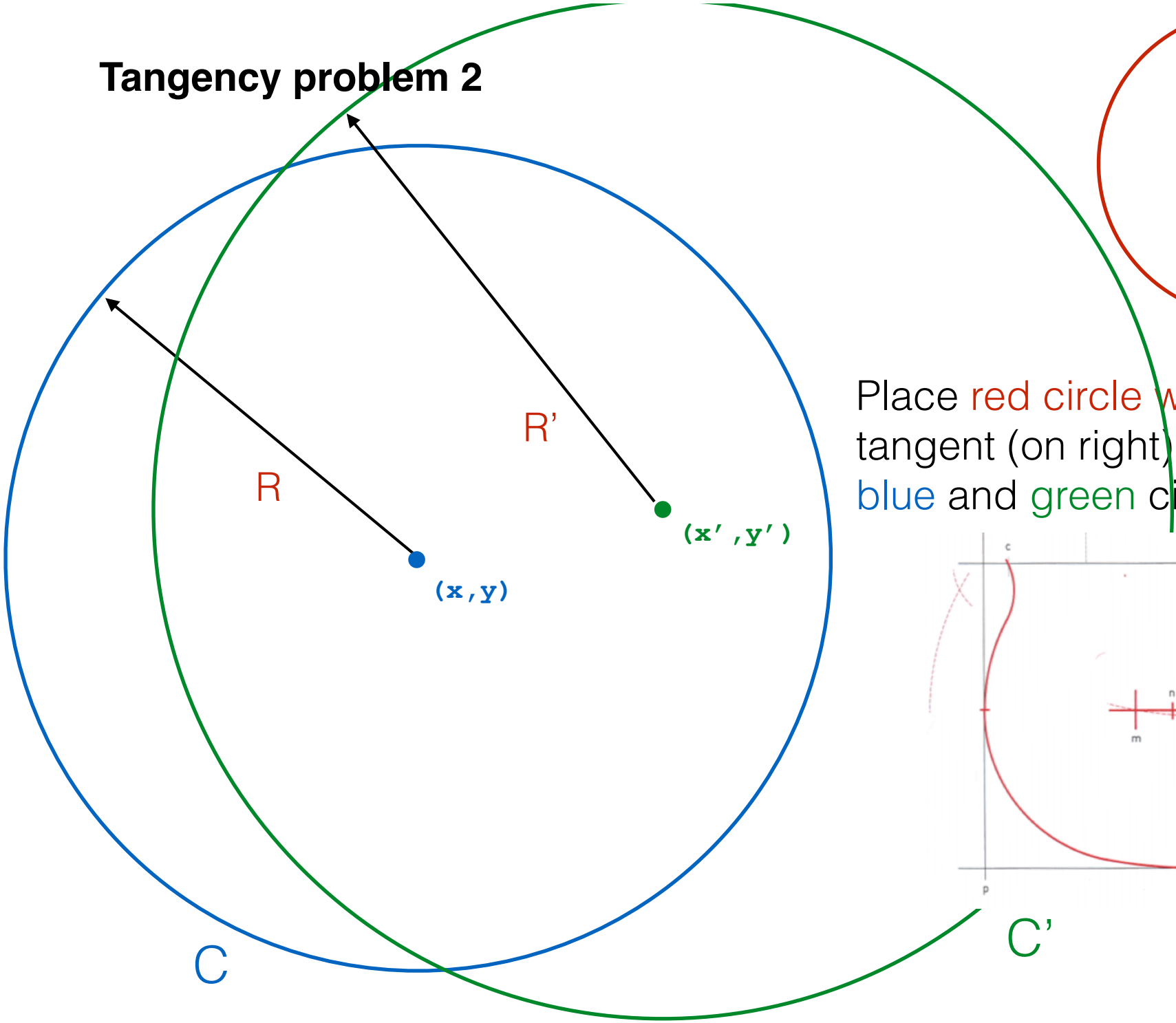

```
(let*
  ((L (vertical (point x y)))
   (C (circle (point x′ y′) R)))

 <what goes in here??>)
```

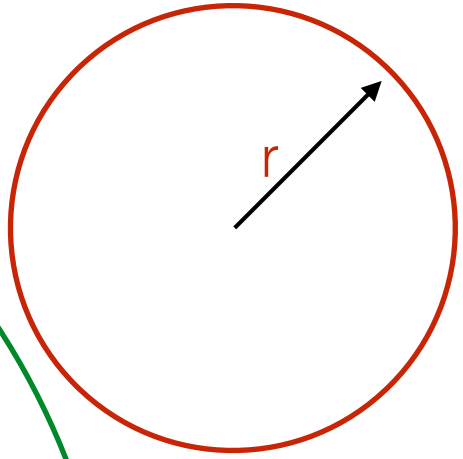
Tangency problem 1



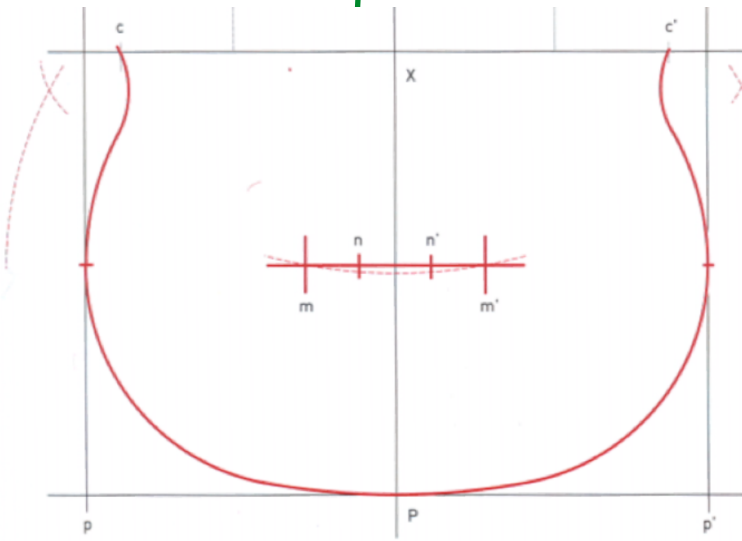

```
(let*
  ((L (vertical (point x y)))
   (C (circle (point x′ y′) R)))
 (circle
   (top (intersect
          (vertical
            (xshift (first-point L) r))
          (circle (center C) (- R r))))
   r))
```

Tangency problem 2



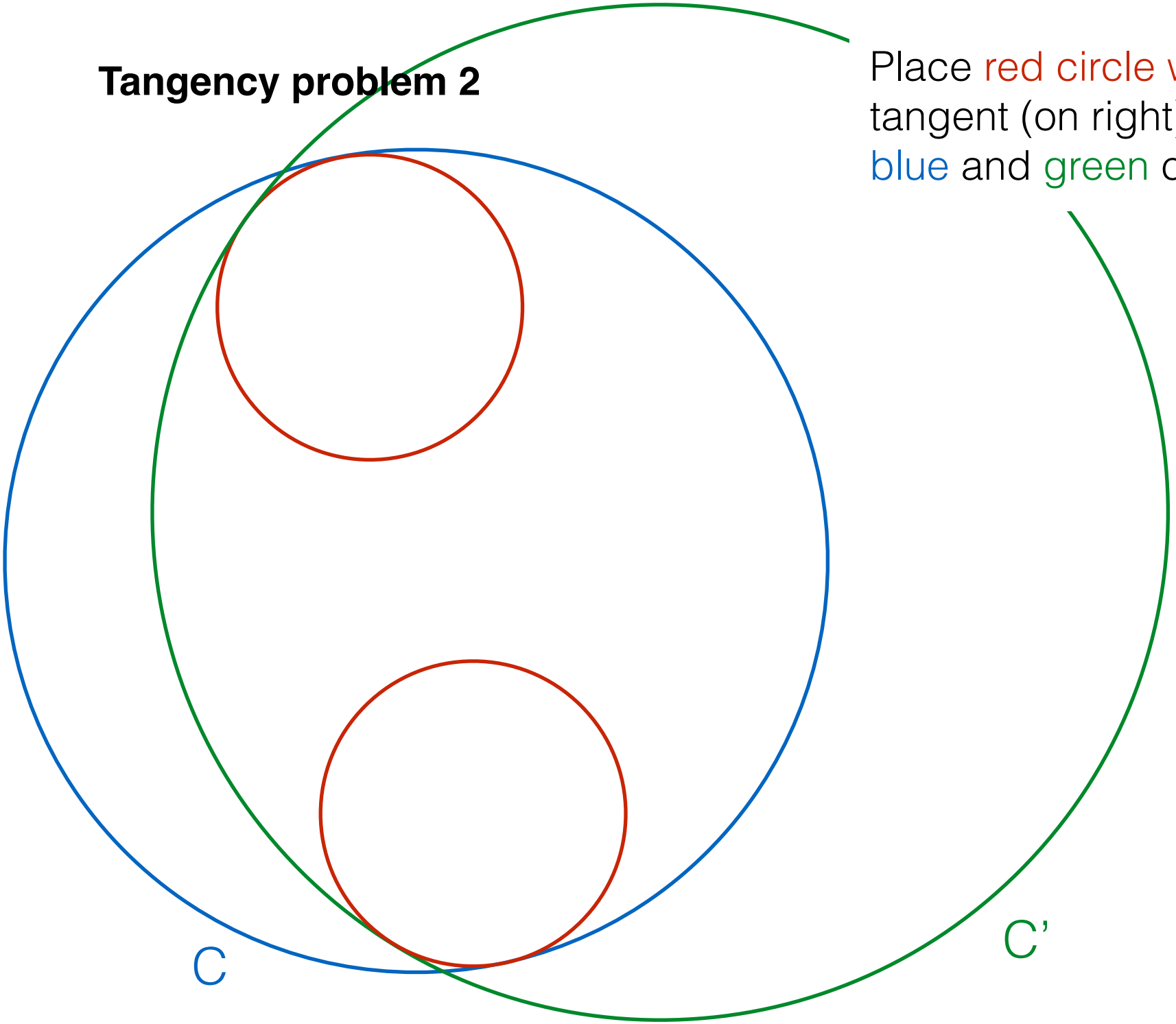


Place red circle with radius r tangent (on right) to *inside* of blue and green circles C, C' .



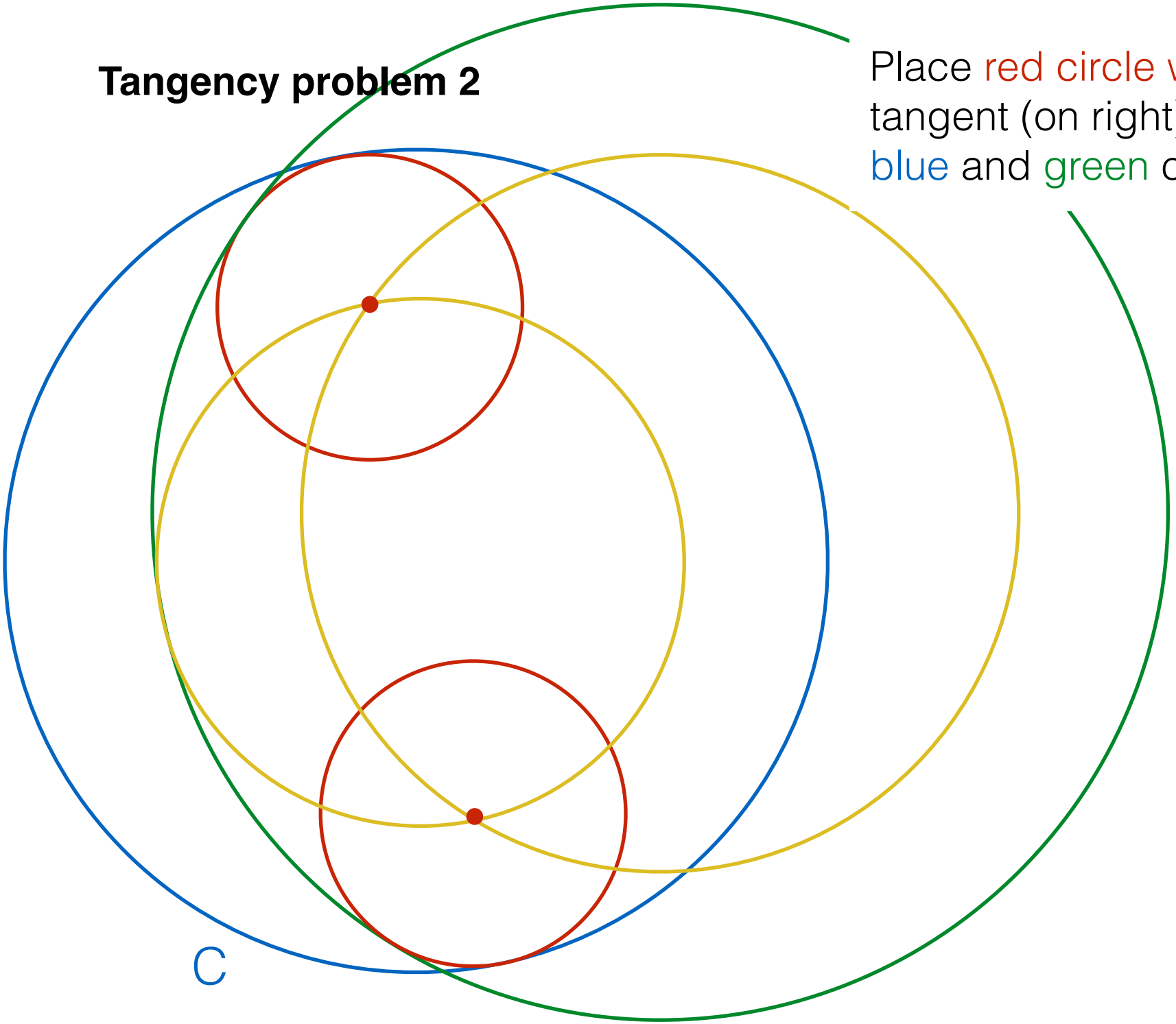
Tangency problem 2

Place red circle with radius r tangent (on right) to *inside* of blue and green circles C, C' .



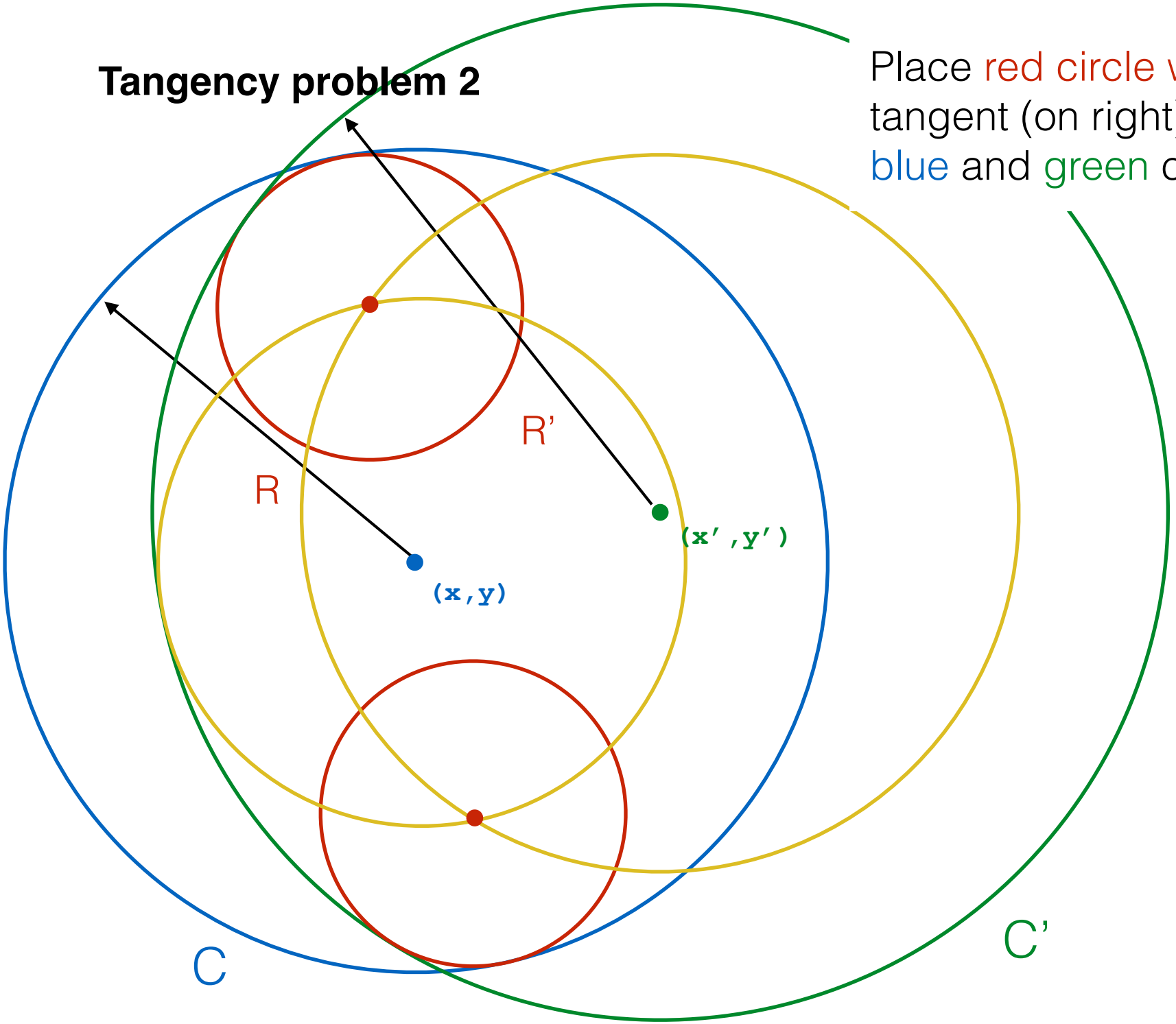
Tangency problem 2

Place red circle with radius r tangent (on right) to *inside* of blue and green circles C, C' .



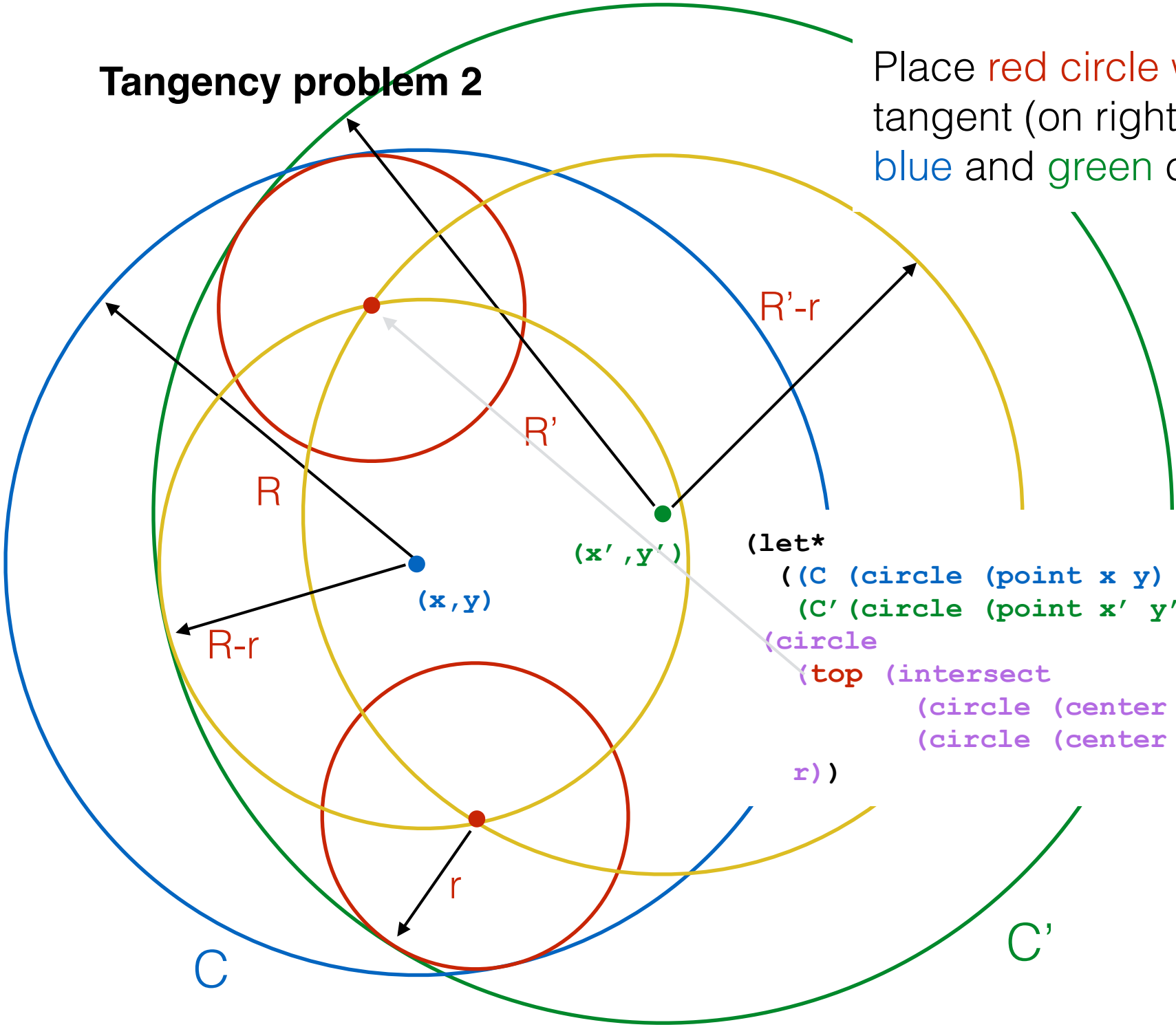
Tangency problem 2

Place red circle with radius r tangent (on right) to *inside* of blue and green circles C, C' .



Tangency problem 2

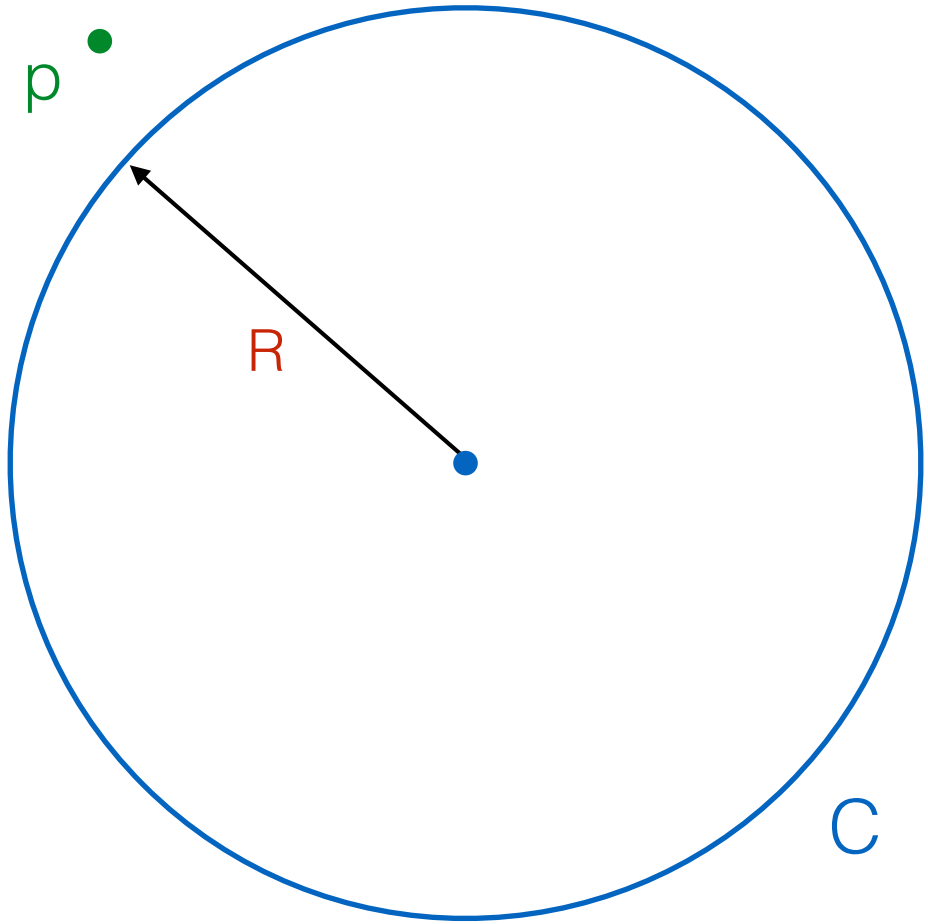
Place red circle with radius r tangent (on right) to *inside* of blue and green circles C, C' .

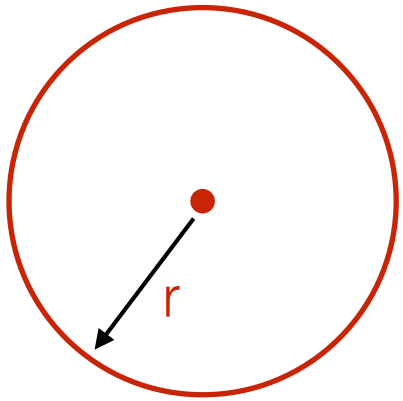


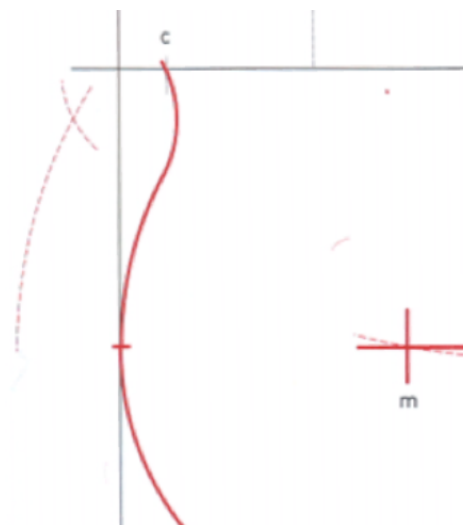

```
(let*
  ((C (circle (point x y) R))
   (C'(circle (point x' y') R')))
 (circle
   (top (intersect
          (circle (center C) (- R r)))
          (circle (center C') (- R' r))))
   r))
```

Tangency problem 3

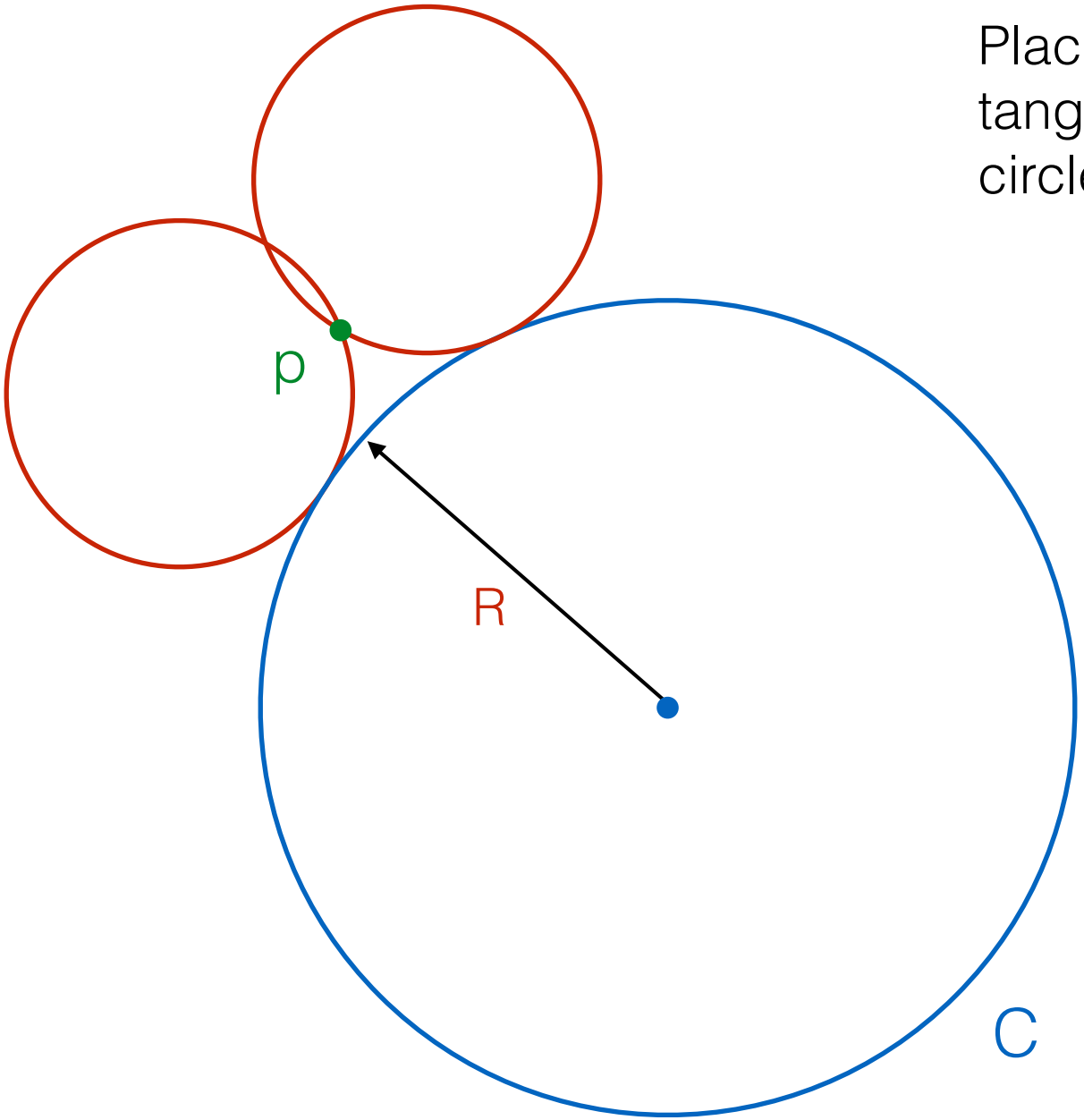
Place red circle with radius r tangent to *outside* of blue circle C and green point p.



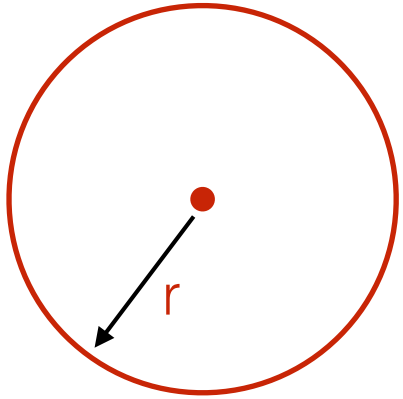


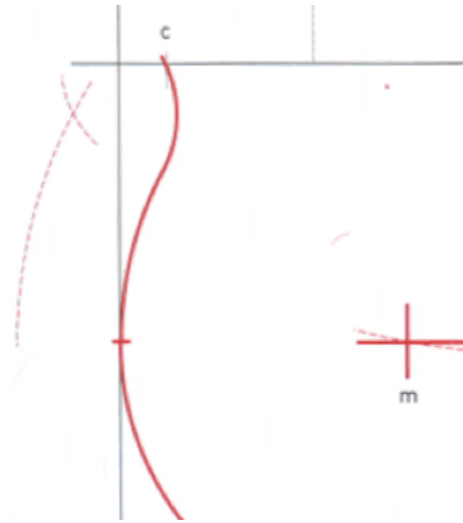


Tangency problem 3

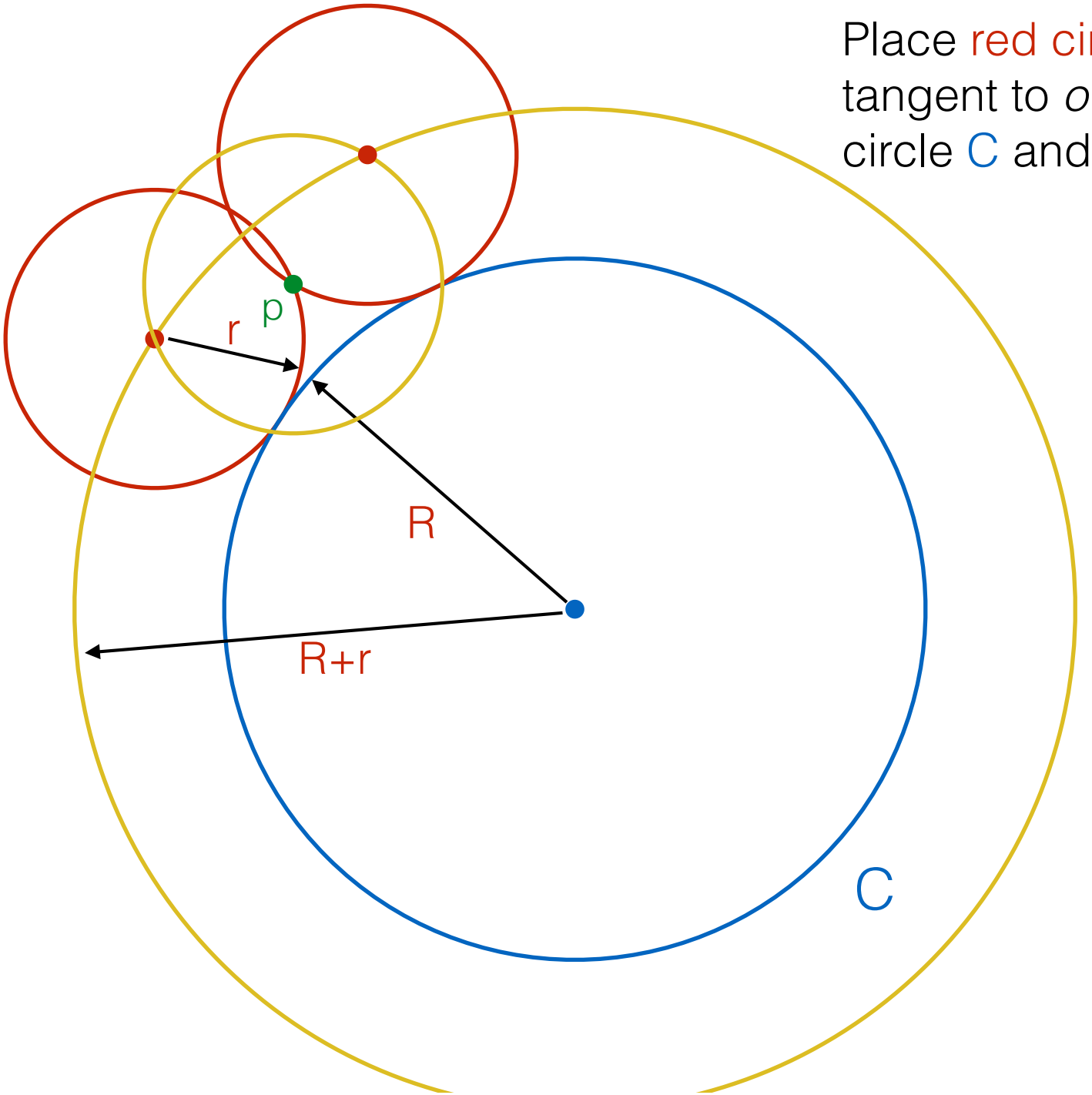


Place red circle with radius r tangent to *outside* of blue circle C and green point p.



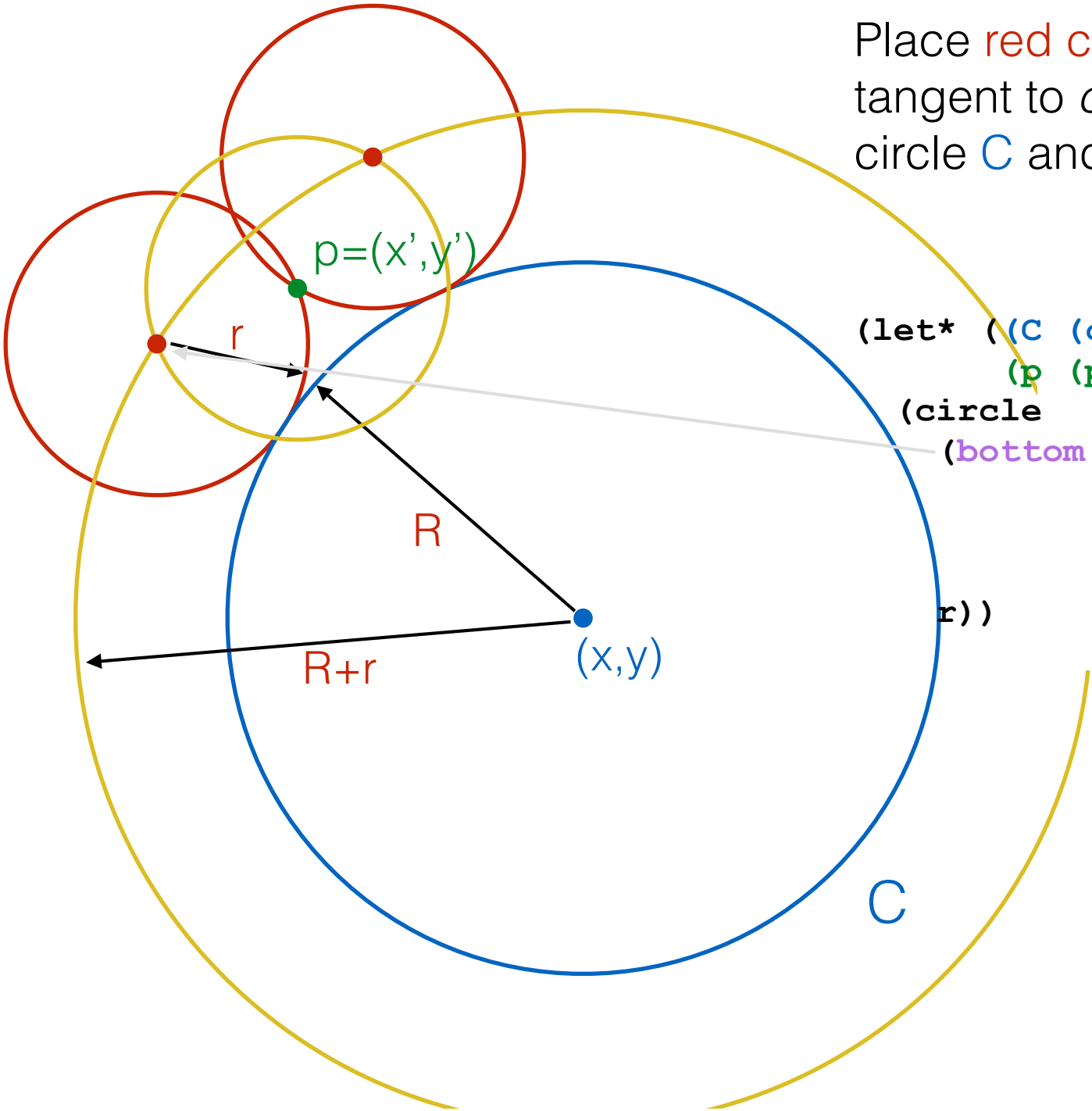


Tangency problem 3



Place red circle with radius r tangent to *outside* of blue circle C and green point p.

Tangency problem 3

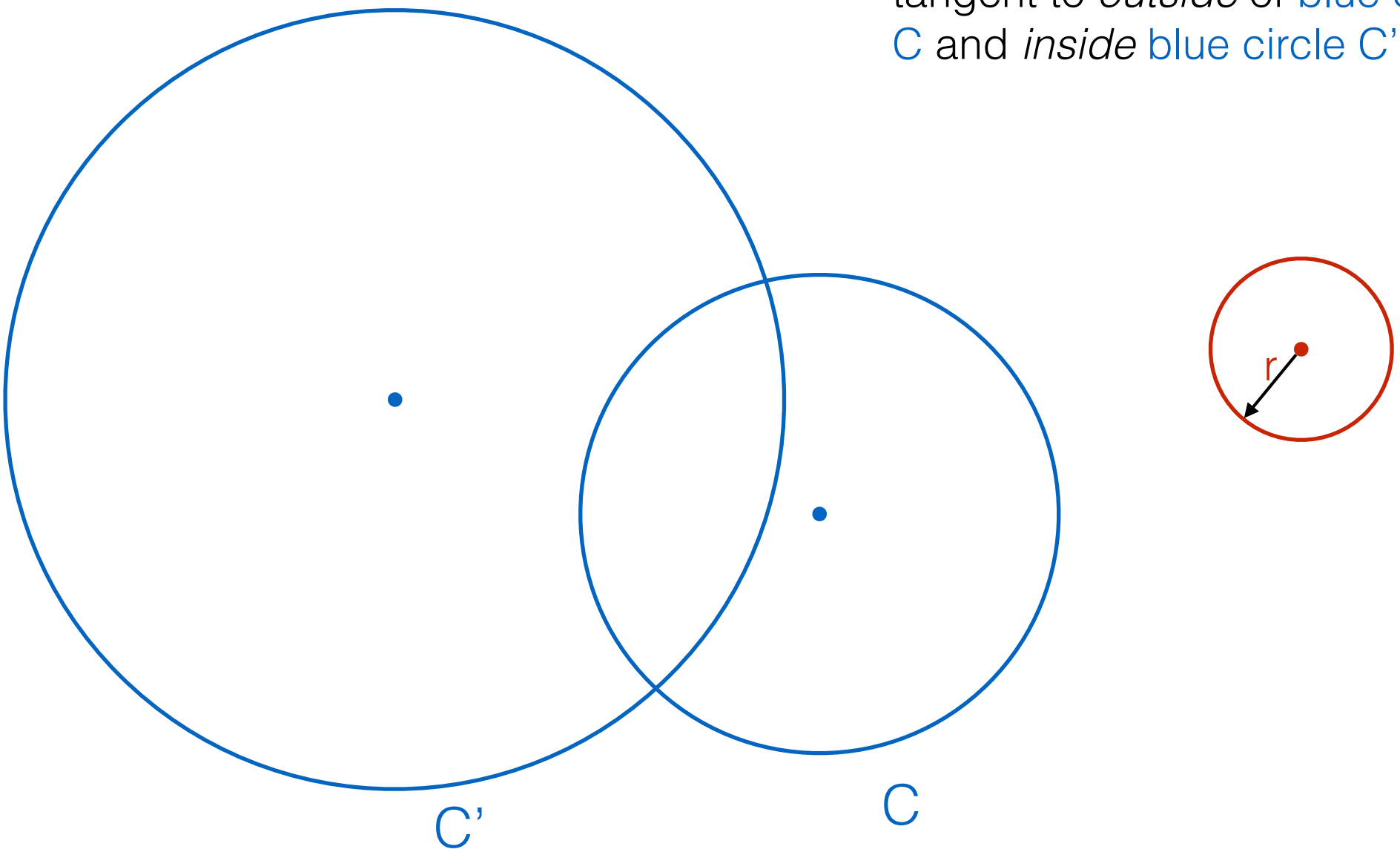


Place red circle with radius r tangent to *outside* of blue circle C and green point p.

```
(let* ((C (circle (point x y) R))
       (p (point x' y')))
  (circle
    (bottom (intersect
              (circle p r)
              (circle (center C)
                      (+ R r))))
    r))
```

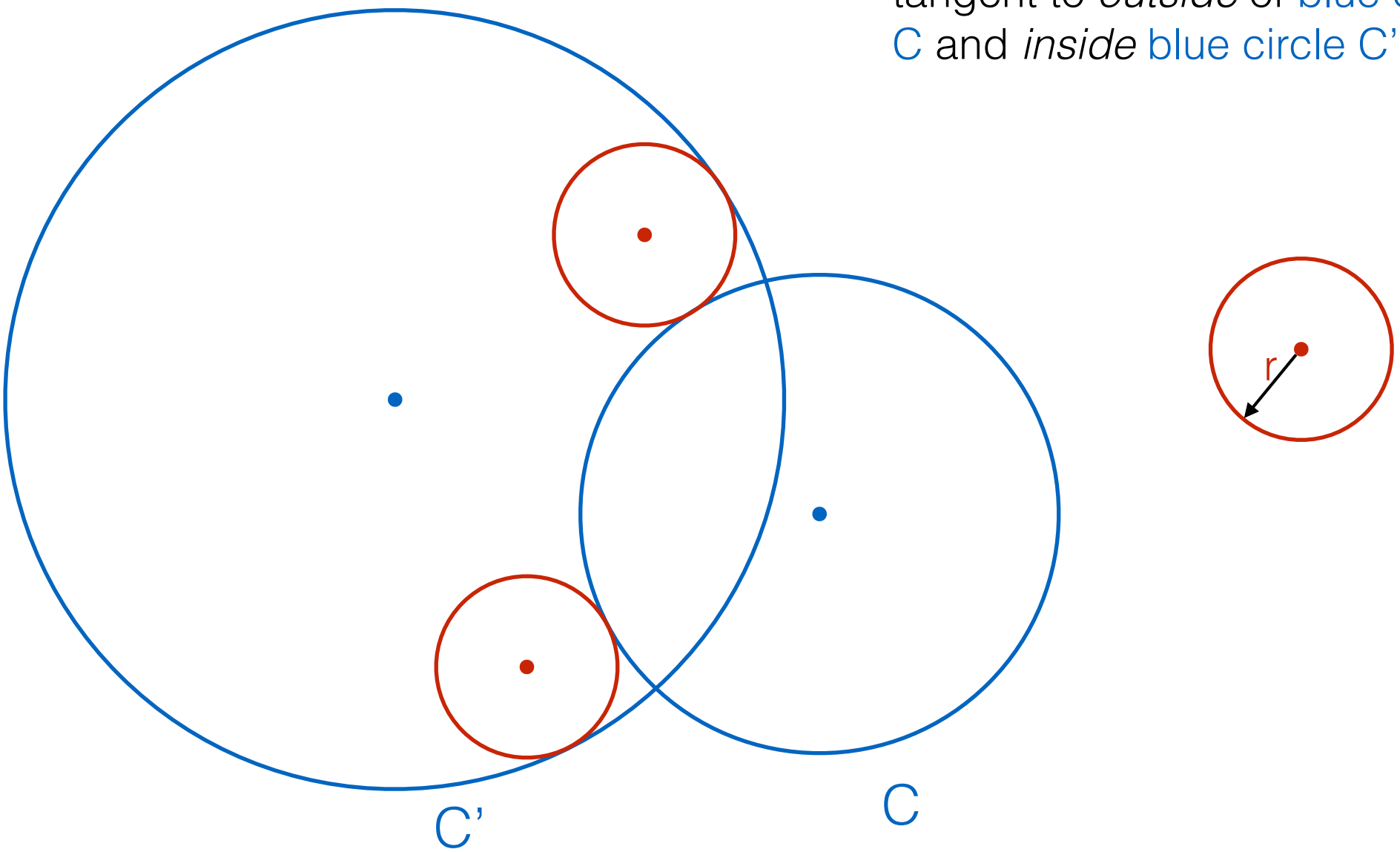
Tangency problem 4 *(they're all the same!)*

Place red circle with radius r tangent to *outside* of blue circle C and *inside* blue circle C'.



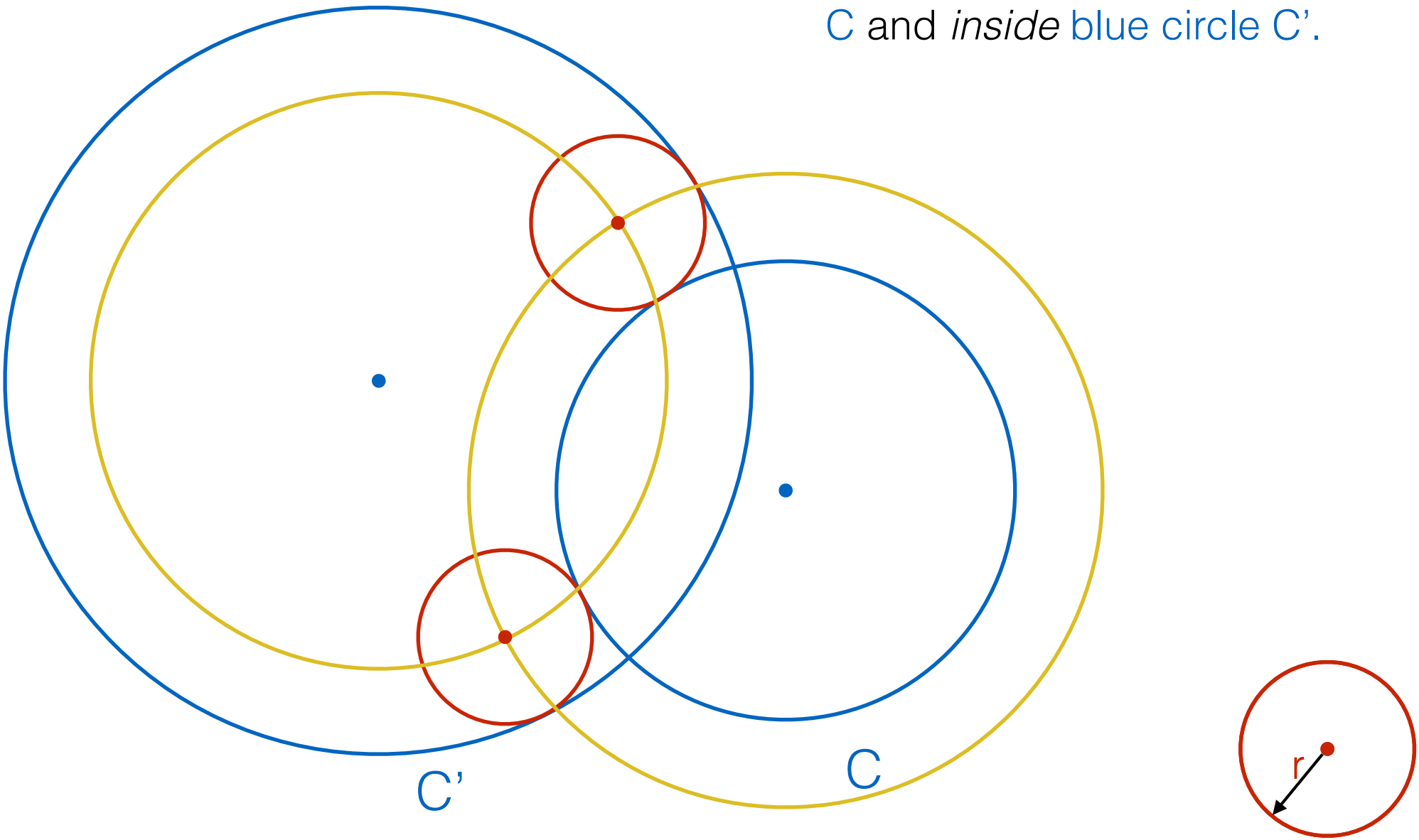
Tangency problem 4 *(they're all the same!)*

Place red circle with radius r tangent to *outside* of blue circle C and *inside* blue circle C'.



Tangency problem 4 *(they're all the same!)*

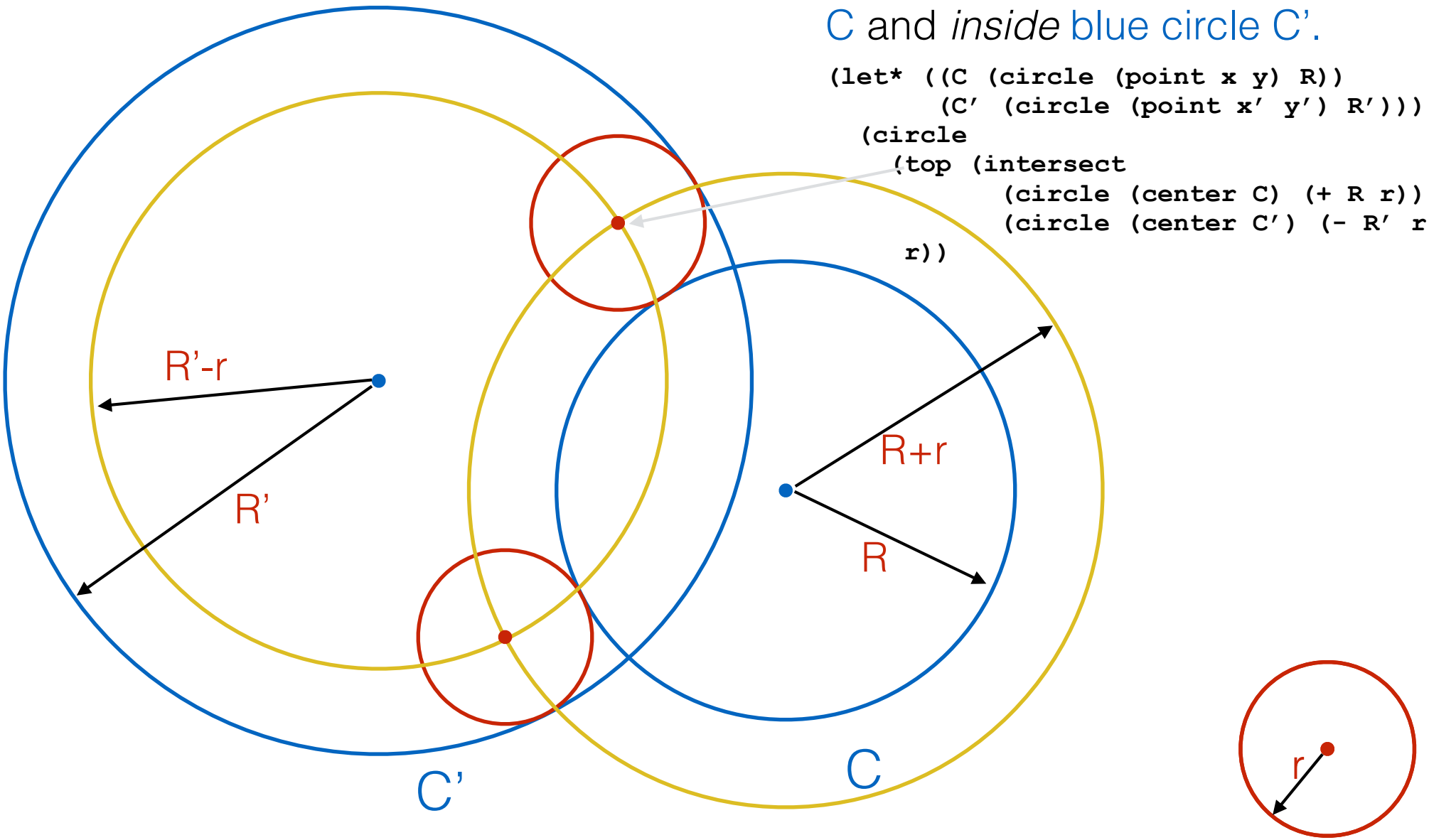
Place red circle with radius r tangent to *outside* of blue circle C and *inside* blue circle C'.



Tangency problem 4 *(they're all the same!)*

Place red circle with radius r tangent to *outside* of blue circle C and *inside* blue circle C'.

```
(let* ((C (circle (point x y) R))
       (C' (circle (point x' y') R')))
  (circle
    (top (intersect
           (circle (center C) (+ R r))
           (circle (center C') (- R' r))))
    r))
```

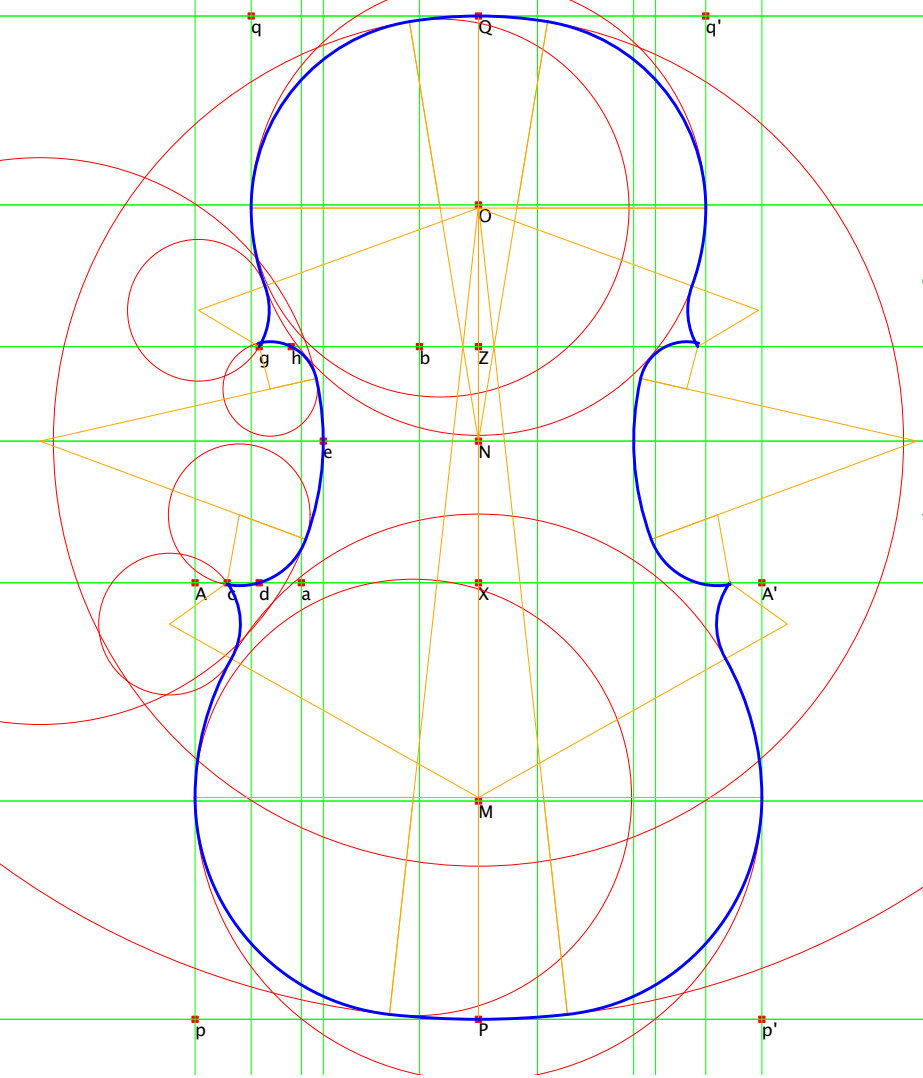


Diversion

Q_1: how close are we to historical reality?
A_1: "up to isomorphism"

Q_2: how can we distinguish $f(x)=(x+1)(x-1)$ from $g(x)=x^2-1$?

A_2: "up to isomorphism"



[Andrea Guarneri alto viola]

~~How do we~~ know in what order arcs are drawn from the framework?

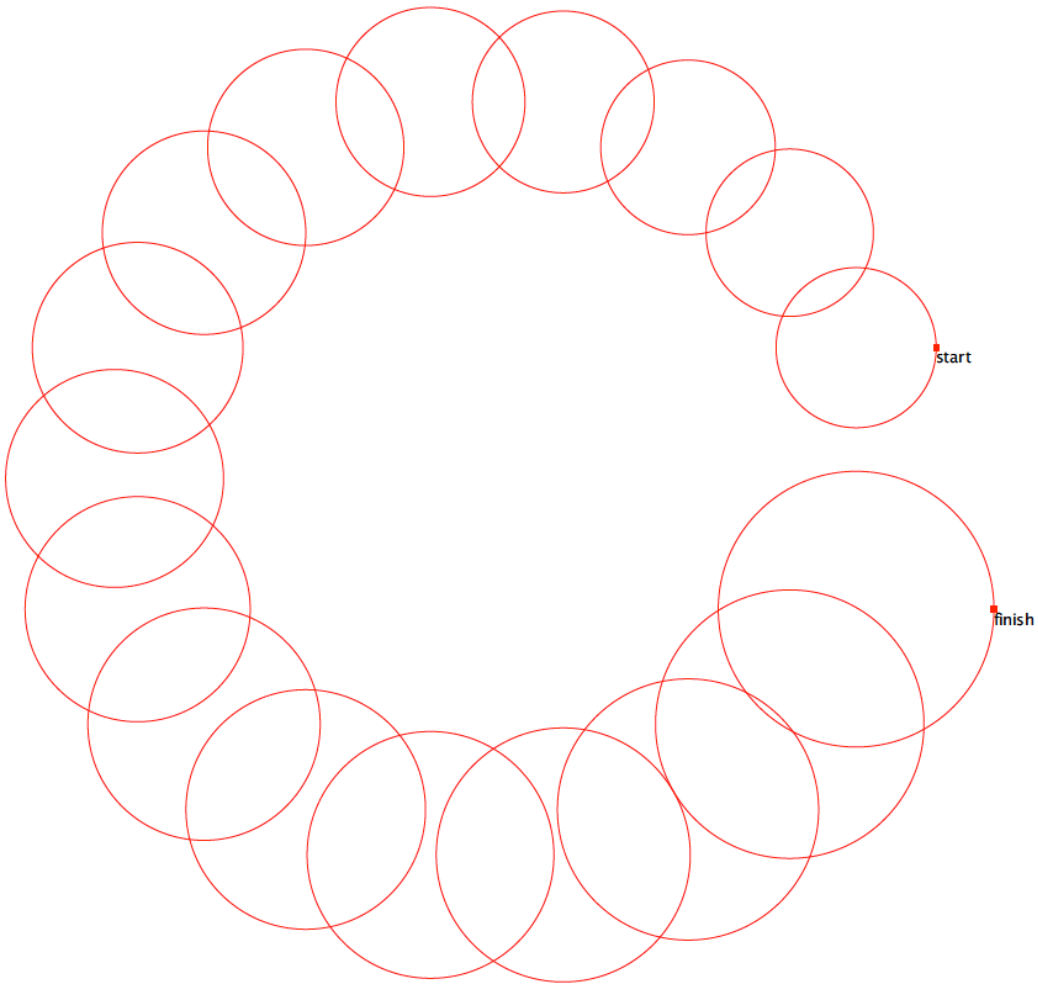
For example, $R1_{upper}$ precedes $R2_{upper}$ and $R3_{upper}$, but which of $R2_{upper}$, $R3_{upper}$ is ~~drawn first~~?

It doesn't really matter—-we equate these ~~using some~~ rationale of *isomorphism* (similar form)… like two software implementations that *do the same thing…*

(make-curve *p q* (list O_1 O_2 … O_n))

Example:

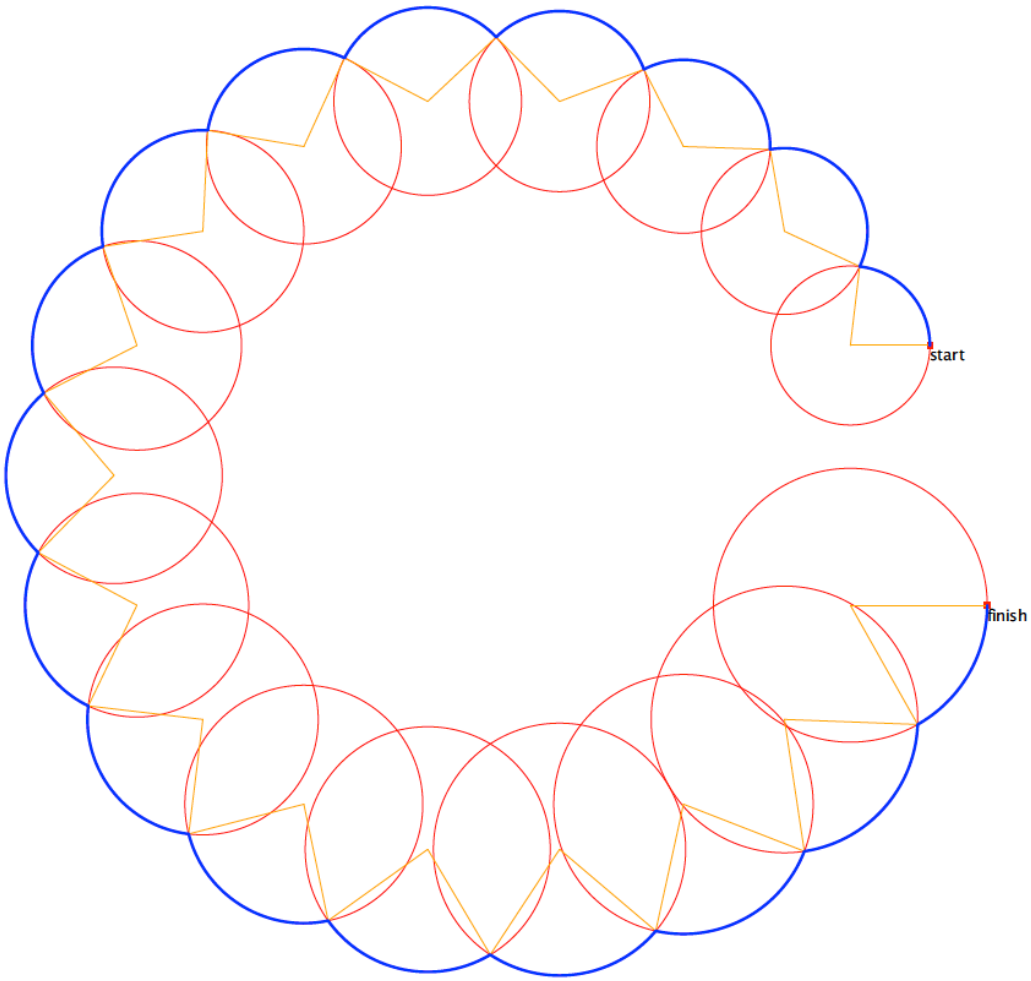
```
(let* ((C list of circles below)
       (p (label "start" (first C)))
       (q (label "finish" (last C))))
  (list p q C))
```



(make-curve *p q* (list O_1 O_2 … O_n))

Example:

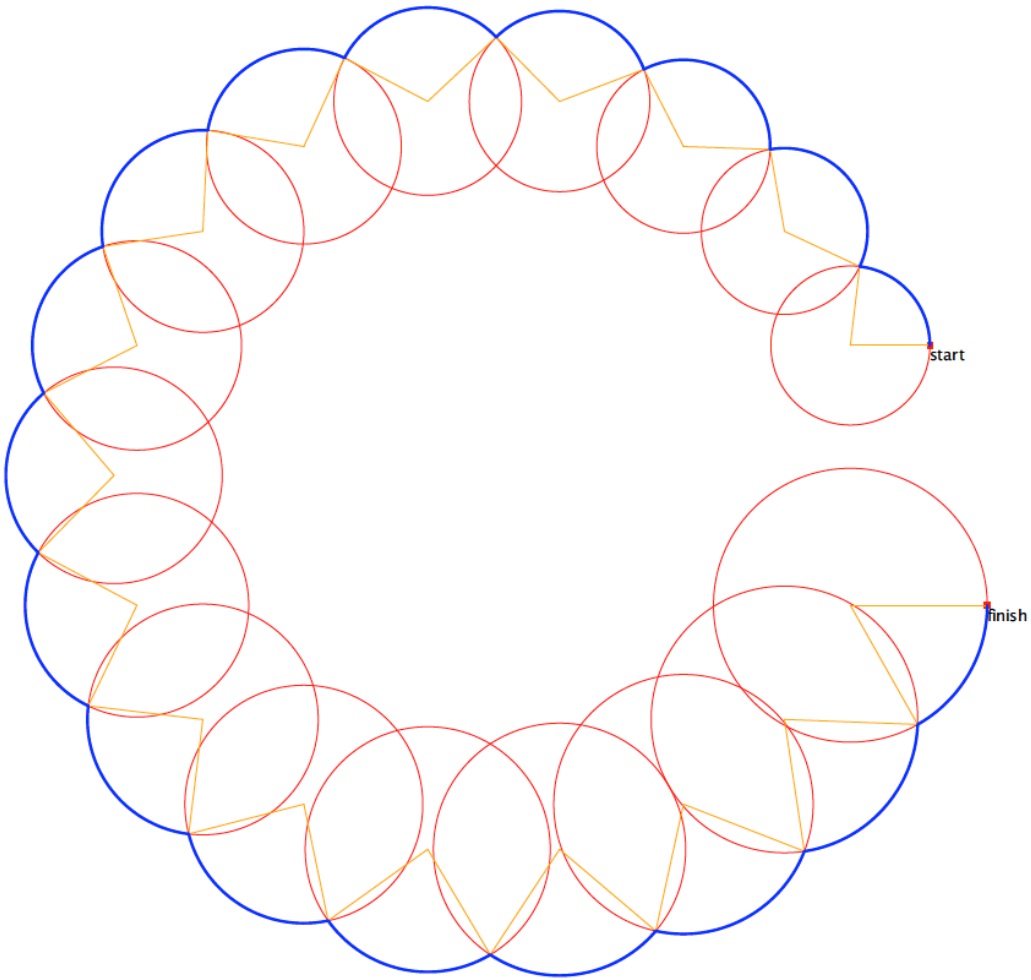
```
(let* ((C list of circles below)
       (p (label "start" (first C)))
       (q (label "finish" (last C))))
  (list p q C (make-curve p q C))
```

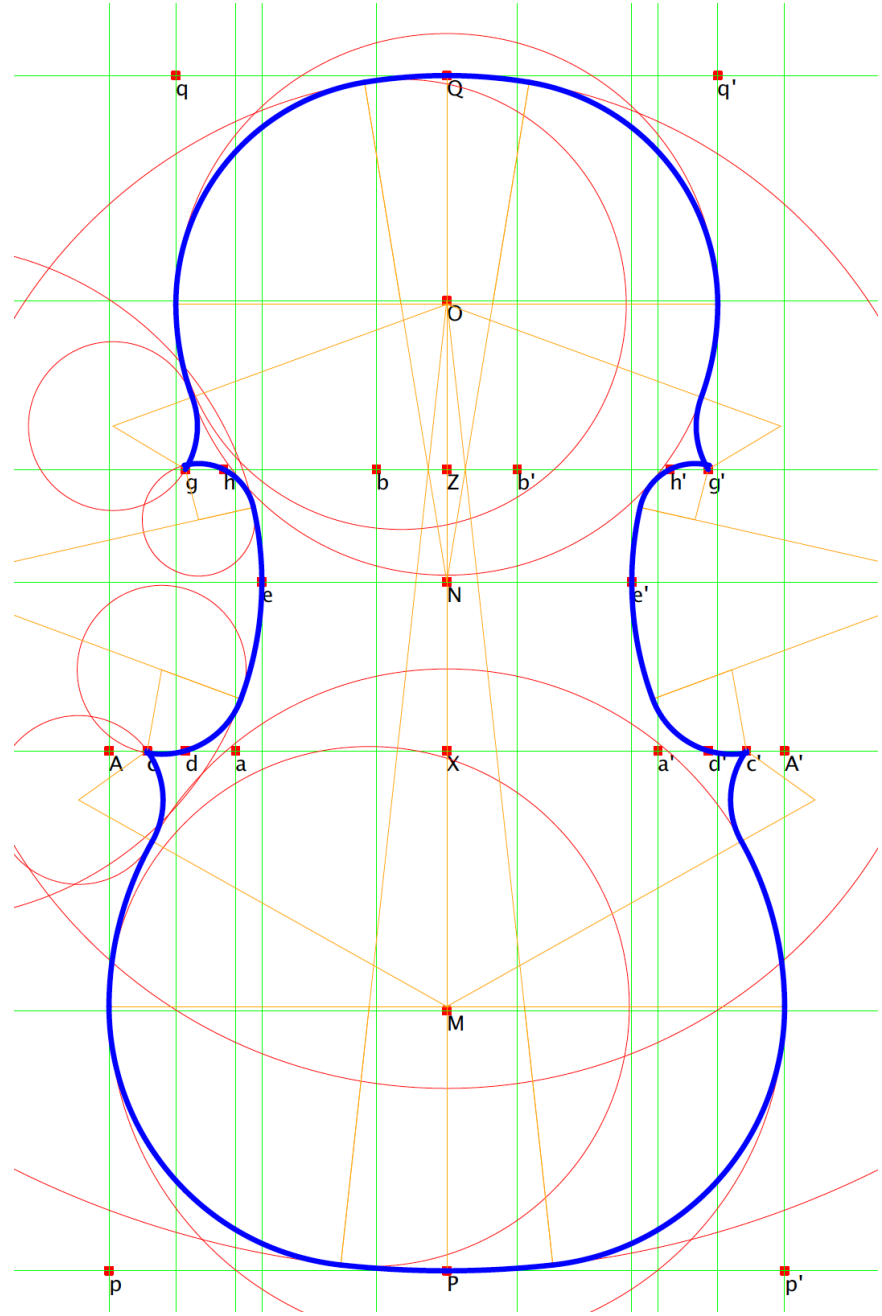


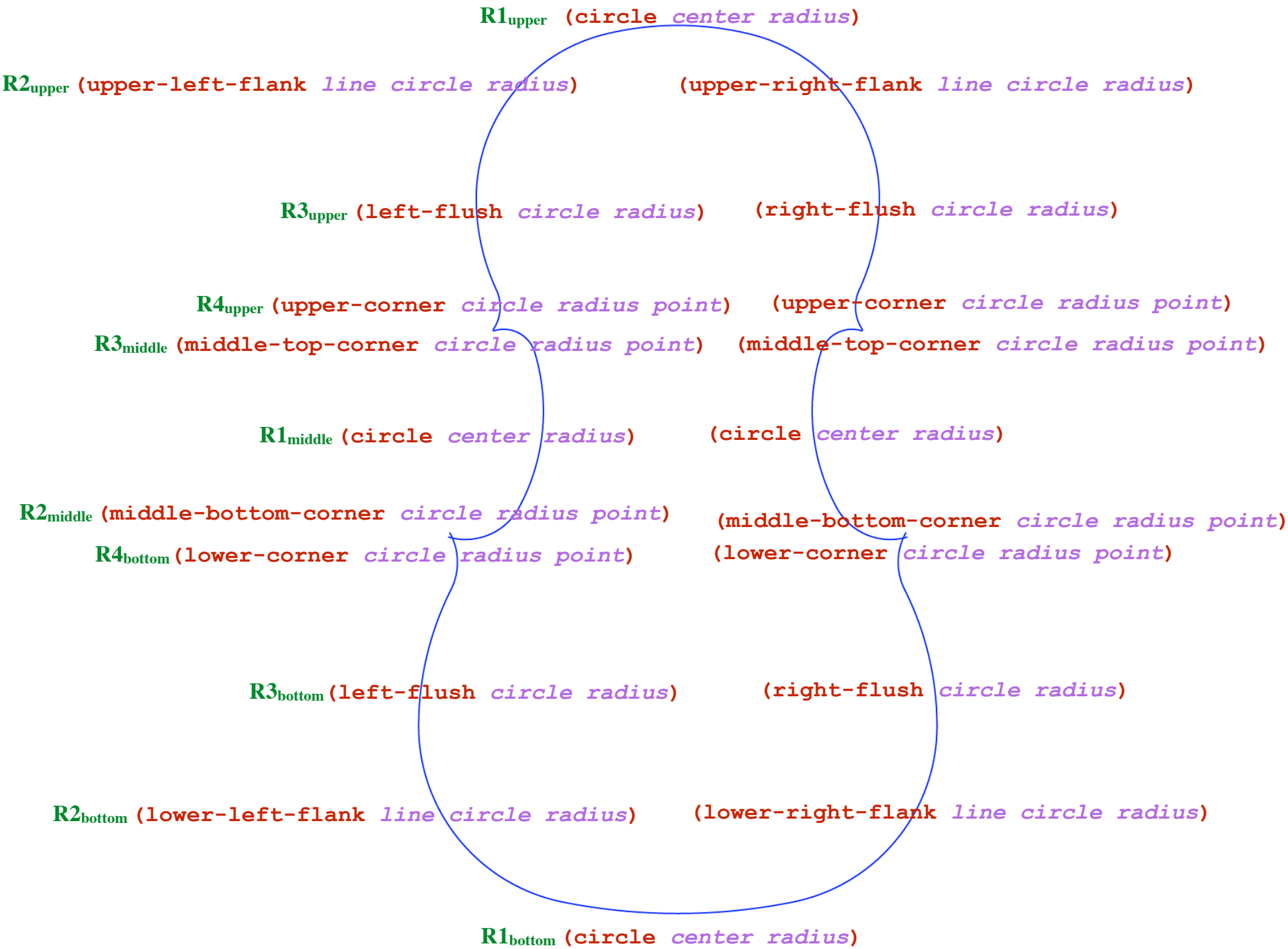
(make-curve *p q* (list O_1 O_2 … O_n))

Example:

```
(let* ((C list of circles below)
       (p (label "start" (first C)))
       (q (label "finish" (last C))))
  (list p q C (make-curve p q C))
```






R1upper (circle center radius)
R2upper (upper-left-flank line circle radius)
(upper-right-flank line circle radius)
R3upper (left-flush circle radius)
(right-flush circle radius)
R4upper (upper-corner circle radius point)
(upper-corner circle radius point)
R3middle (middle-top-corner circle radius point)
(middle-top-corner circle radius point)
R1middle (circle center radius)
(circle center radius)
R2middle (middle-bottom-corner circle radius point)
(middle-bottom-corner circle radius point)
R4bottom (lower-corner circle radius point)
(lower-corner circle radius point)
R3bottom (left-flush circle radius)
(right-flush circle radius)
R2bottom (lower-left-flank line circle radius)
(lower-right-flank line circle radius)
R1bottom (circle center radius)

For example…

$R2_{upper}$ `(upper-left-flank line circle radius)`

sitting in the geometry engine is… (not that you need to know)…

```
(define (upper-left-flank L C r)
  (circle
    (top (intersect
           (vertical
             (xshift (first-point L) r))
           (circle (center C) (- R r))))
    radius))
```

The point is to make the code you're writing as readable as possible, eliminating detail that you don't want to see…

File format describing an outline

```
#lang racket

(require “Geometry-Engine.rkt")  ; in the same folder/directory as this file…


(elaboration #t)       ; draw extra elaboration noted
(mirroring #t)         ; mirror bass/treble (so you don’t have to do stuff twice
(arcthickness 2)       ; thickness of the outline
(arccolor “blue")      ;   and its color

(coded-by “Harry Mairson”)
(title "Violoncello by Antonio Stradivari ['Cristiani', 1700]")

(define (Cristiani)
  (let* (
         ; FRAMEWORK goes here
         (v1 e1)
         (v2 e2)
           …
         ; UPPER BOUT
         ; MIDDLE BOUT      (in any order, but after the framework)
         ; LOWER BOUT
        )
    (list (make-curve c P (list …))  ; to draw the three bouts
          (make-curve Q g (list …))
          (make-curve g c (list …))
          ; and whatever else you would like to elaborate…
      )))

(sketch (Cristiani))

(end-drawing)
```

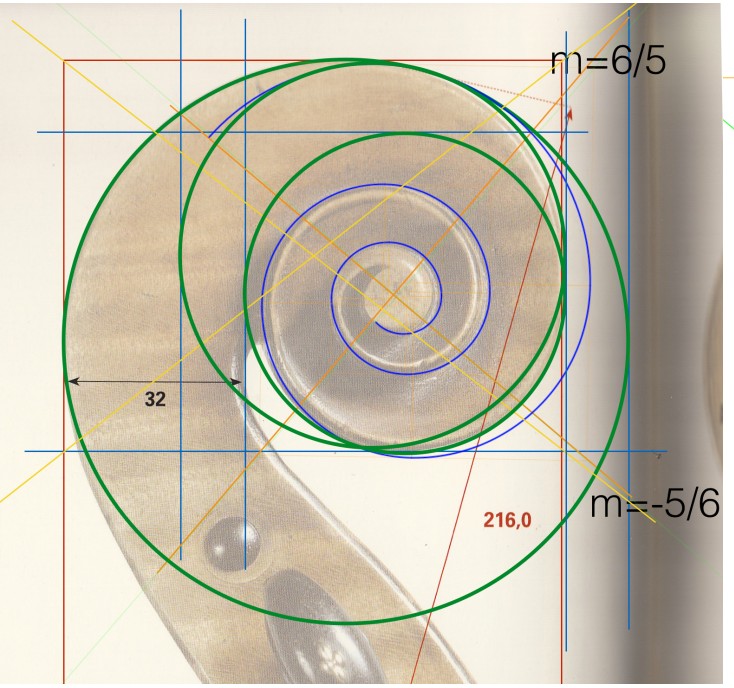
```
#lang racket

; This file is a TEMPLATE for the exercise of drawing the Amati violin in "Traité de Lutherie" (pages 114–129).
; The code has a variety of places labelled "..." where you should try plugging in the right expressions.
; What you need to do is look at the description in the book, and then translate it into code.

; There's a "solution sheet" that you can see in the file Amati.rkt, but the exercise is to try filling in the
; blanks without looking.

; HERE'S HOW TO DO IT:

; Notice that the semicolon ";" serves to make certain text comments to the reader (yellow-brown typeface).
; Fill in the ...s from top to bottom.  If you run the file as is, you'll get a PDF drawing with the point
; X only.  Then put A in place, AND "uncover" the A in the parts list by moving it to the other side of the ;
; mark.  Then you'll have X and A.  And so on to complete the drawing.

; You have to go back and forth, filling in the definitions, and running the file to see the output.  If the
; output looks good, go on.   If it looks bad, you need to fix it.  With a compass and straightedge, you'd be
; erasing and redrawing.  Here' you debug your code, and press "Run" to redraw.
```

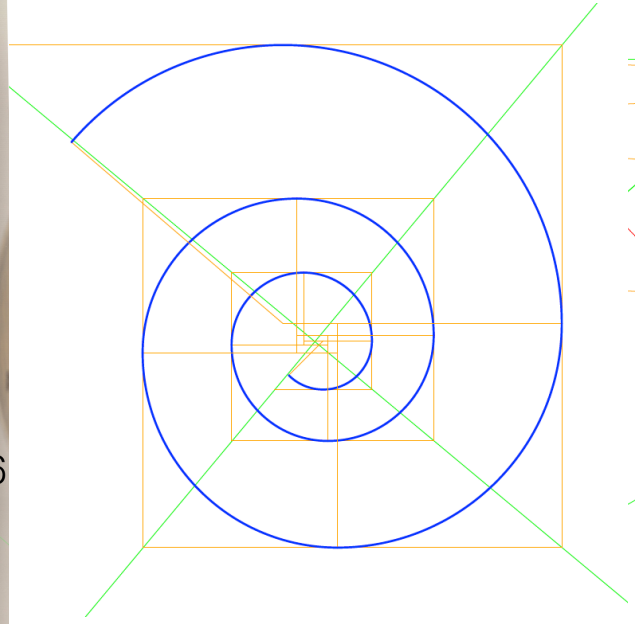
```
#lang racket

…

(require "Geometry-Engine.rkt")

(elaboration #t)
(mirroring #t)
(arcthickness 1)
(arccolor "blue")

(title "Violin by Andrea Amati (const. François Denis)")
; Violin by Andrea Amati

(define (Amati)

; LAYOUT OF THE AREA on which the curves are drawn...

(let* ((xq 208) ;; 208mm in the Amati
       (X (label "X" (point 0 0)))
;       (A (label "A" ...))
;       (Q (label "Q" ...))
;       (N (label "N" ...))
;       (q (label "q" ...))
;       (qp (label "q'" (mirror q)))  ; WHEN YOU "UNCOVER" q, UNCOVER THIS TOO...
;       (O (label "O" ...))
;       (Z (label "Z" ...))
;       (P (label "P" ...))
;       (p (label "p" ...))
;       (pp (label "p'" (mirror p)))  ; WHEN YOU "UNCOVER" p, UNCOVER THIS TOO...
;       (M (label "M" ...))
;       (a (label "a" ...))
;       (b (label "b" ...))
;       (e (label "e" ...))
;       (c (label "c" ...))
;       (d (label "d" ...))
;       (h (label "h" ...))
;       (g (label "g" ...)
```

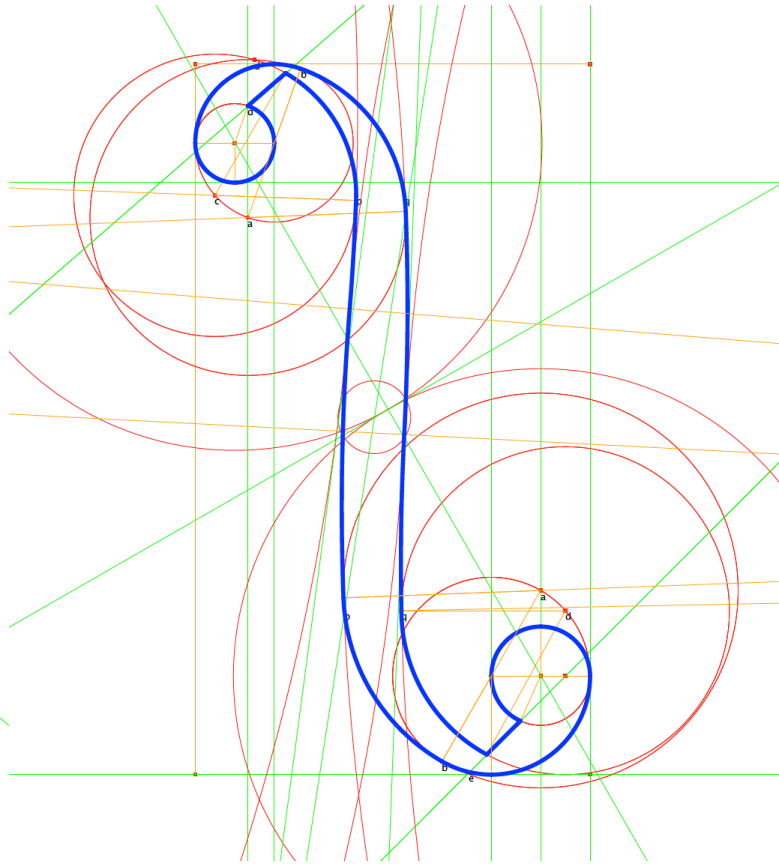
```
; ***** DRAWING THE OUTLINE *****

; THE LOWER BOUTS...
;       (R1lower (circlefrom ...))                ; lower block
;       (R2lower (lower-left-flank ...))          ; lower left flank
;       (R3lower (left-flush ...))                ; above R2lower, towards corner
;       (R4lower (lower-corner ...))              ; corner
;       (lower-curve (make-curve ...start, end points... (list ...circles making curve go here)))
                                                  ; these curves, put together

; THE UPPER BOUTS...
;       (R1upper (circlefrom ...))                ; top block
;       (R2upper (upper-left-flank ...))          ; upper left flank and corner (why the Amati looks so round...)
;       (R3upper (upper-corner ...))              ; corner
;       (upper-curve (make-curve ...start, end points... (list ...circles making curve go here)))
                                                  ; these curves, put together

; THE MIDDLE BOUTS...
;       (f (label "f" (xshift e (- (distance X Z)))))
;       (R1middle (circlefrom ...))               ; main curve in the C-bout
;       (R2middle (middle-top-corner ...))        ; top arc in curve towards upper corner
;       (R3middle (middle-bottom-corner ...))     ; bottom arc in curve towards lower corner
;       (middle-curve (make-curve ...start, end points... (list ...circles making curve go here))
       )

 ; Parts list (to be drawn)
 (list X ; A Q N q qp O Z P p pp M a b e c d h g
       ; (horizontal N) (horizontal O) (horizontal Z)
       ; (horizontal P) (horizontal Q) (horizontal X) (horizontal M)
       ; (vertical p) (vertical q) (vertical b) (vertical e) (line p (mirror q))

       ; R1lower R2lower R3lower R4lower
       ; R1upper R2upper R3upper
       ; f R1middle R2middle R3middle

       ; upper-curve lower-curve middle-curve

             )))

(sketch (Amati))

(end-drawing)
```

Coming events?

A better language description… or… ??More research on instrument evolution (Gofriller, Bergonzi/Montagnana, …)
Useful tools for other parts of design and construction…
Automated design synthesis







see Denis 2.2, “The turn of the scroll”
(“Drôle de tête”)

also Denis, course notes…