1. Preliminaries

$$
\begin{array}{l}
\text{\{-\# OPTIONS -W \#-\}} \\
\text{\{-\# LANGUAGE TypeSynonymInstances, FlexibleInstances, Rank2Types \#-\}} \\
\textbf{module}\ \mathit{Crossing}\ \textbf{where} \\
\textbf{import}\ \mathit{Control.Monad.State} \\
\textbf{import}\ \mathit{Data.Monoid}\ (\mathit{Monoid}\ (..)) \\
\textbf{import}\ \mathit{Numeric}\qquad (\mathit{showFFloat})
\end{array}
$$

We define a data type isomorphic to pairs, to express that one thing is above another spatially. For example, two threads might grow rightward in parallel, one above the other.

$$\textbf{data}\ \genfrac{}{}{0pt}{}{\mathit{above}}{\mathit{below}} = \genfrac{}{}{0pt}{}{\mathit{above}}{\mathit{below}}\ \textbf{deriving}\ \mathit{Show}$$

Given a list of things (such as threads ordered from top to bottom), we often want to pick out one or two of them by index and change those few while leaving the rest intact.

$$
\begin{array}{l}
\mathit{at} :: \mathit{Int} \rightarrow ([a] \rightarrow [a]) \rightarrow ([a] \rightarrow [a]) \\
\mathit{at}\ i\ f\ \mathit{xs} = \mathit{ys} + \!\!\! + f\ \mathit{zs}\ \textbf{where}\ (\mathit{ys}, \mathit{zs}) = \mathit{splitAt}\ i\ \mathit{xs} \\[1ex]
\mathit{at1} :: \mathit{Int} \rightarrow (a \rightarrow a) \rightarrow ([a] \rightarrow [a]) \\
\mathit{at1}\ i\ f = \mathit{at}\ i\ (\lambda(x : \mathit{zs}).\, f\ x : \mathit{zs}) \\[1ex]
\mathit{at2} :: \mathit{Int} \rightarrow (\genfrac{}{}{0pt}{}{a}{a} \rightarrow \genfrac{}{}{0pt}{}{a}{a}) \rightarrow ([a] \rightarrow [a]) \\
\mathit{at2}\ i\ f = \mathit{at}\ i\ (\lambda(x : y : \mathit{zs}).\, \textbf{let}\ \genfrac{}{}{0pt}{}{x'}{y'} = f\ \genfrac{}{}{0pt}{}{x}{y}\ \textbf{in}\ x' : y' : \mathit{zs})
\end{array}
$$

2. Weaving

Weaving is basically the process of taking a list of crossings and following those commands to permute a list of threads. A crossing is basically an command to swap two adjacent threads, so the two threads to swap are identified by a single integer index into the list of threads. A crossing is either positive (right-handed, ⤭) or negative (left-handed, ⤮).

$$\textbf{data}\ \mathit{PN} = \text{⤭} \mid \text{⤮}\ \textbf{deriving}\ \mathit{Show}$$

Each thread records the history of how it has been swapped with other threads. The history is a list of *Signal*s, one per swap experienced by the thread (from most recent to most ancient—that is, in reverse order).

$$
\begin{array}{l}
\textbf{data}\ \mathit{Signal}\ \ c\ a\ \ = \mathit{Signal}\ \ \mathit{OU}\qquad c\ a\ \ \textbf{deriving}\ \mathit{Show} \\
\textbf{data}\ \mathit{Thread}\ c\ a\ t = \mathit{Thread}\ t\ [\mathit{Signal}\ c\ a]\ \textbf{deriving}\ \mathit{Show} \\[1ex]
\textbf{instance}\ \mathit{Functor}\ (\mathit{Thread}\ c\ a)\ \textbf{where}\ \mathit{fmap}\ f\ (\mathit{Thread}\ t\ \mathit{ss}) = \mathit{Thread}\ (f\ t)\ \mathit{ss}
\end{array}
$$

The most important thing that a *Signal* records is whether the thread crossed *under* another thread (⇢) or not (→). This information has type OU. At each crossing, one thread goes over the other, depending on the sign of the crossing.

$$
\begin{array}{l}
\textbf{data}\ \mathit{OU} = \rightarrow \mid \dashrightarrow\ \textbf{deriving}\ (\mathit{Eq}, \mathit{Show}) \\[1ex]
\mathit{ou}\nearrow, \mathit{ou}\searrow :: \mathit{PN} \rightarrow \mathit{OU} \\
\mathit{ou}\nearrow\ \text{⤭} = \dashrightarrow \\
\mathit{ou}\nearrow\ \text{⤮} = \rightarrow
\end{array}
$$

$$
\begin{aligned}
&\mathit{ou}{\searrow}\ \text{⤭} = \rightarrow \\
&\mathit{ou}{\searrow}\ \text{⤮} = \dashrightarrow
\end{aligned}
$$

A crossing can also contain information such as its location and the tangent directions of the two threads. Between the histories of the two threads, the location of the crossing is shared whereas the tangent directions are split. In other words, each *Signal* records one crossing location (the same between the two threads, of type c) and one tangent direction (different between the two threads, of type a).

Actually, weaving operates not on a list of *Thread*s but on a list of *Maybe Thread*s. The reason is that sometimes we want to imagine that a thread is there (and swap it with other threads) but not draw it yet. We represent such an imaginary thread by *Nothing*. When a real thread crosses an imaginary thread, the real thread always goes over ($\rightarrow$), so no gap is drawn. (The first state component below, of type c, is explained shortly with *advance*.)

$$
\begin{array}{l}
\mathbf{type}\ M\ c\ a\ t = \mathit{State}\ (c, [\mathit{Maybe}\ (\mathit{Thread}\ c\ a\ t)]) \\
\mathit{crossing} :: \mathit{Monoid}\ c \Rightarrow \mathit{Int} \rightarrow \mathit{PN} \rightarrow c \rightarrow {}^{a}_{a} \rightarrow M\ c\ a\ t\ () \\
\mathit{crossing}\ i\ \mathit{pn}\ c\ {}^{a\nearrow}_{a\searrow} = \mathit{modify}\ (\lambda(c_0, \mathit{threads}).\, (c_0, \mathit{at2}\ i\ (f\ c_0)\ \mathit{threads}))\ \mathbf{where} \\
\quad \begin{array}{lll}
f\ c_0 & \begin{array}{l}\mathit{Just}\ (\mathit{Thread}\ t{\searrow}\ \mathit{ss}{\searrow}) \\ \mathit{Just}\ (\mathit{Thread}\ t{\nearrow}\ \mathit{ss}{\nearrow})\end{array} = & \begin{array}{l}\mathit{Just}\ (\mathit{Thread}\ t{\nearrow}\ (\mathit{Signal}\ (\mathit{ou}{\nearrow}\ \mathit{pn})\ (c_0 \diamond c)\ a{\nearrow} : \mathit{ss}{\nearrow})) \\ \mathit{Just}\ (\mathit{Thread}\ t{\searrow}\ (\mathit{Signal}\ (\mathit{ou}{\searrow}\ \mathit{pn})\ (c_0 \diamond c)\ a{\searrow} : \mathit{ss}{\searrow}))\end{array} \\
f\ c_0 & \begin{array}{l}\mathit{Just}\ (\mathit{Thread}\ t{\searrow}\ \mathit{ss}{\searrow}) \\ \mathit{Nothing}\end{array} = & \begin{array}{l}\mathit{Nothing} \\ \mathit{Just}\ (\mathit{Thread}\ t{\searrow}\ (\mathit{Signal} \rightarrow (c_0 \diamond c)\ a{\searrow} : \mathit{ss}{\searrow}))\end{array} \\
f\ c_0 & \begin{array}{l}\mathit{Nothing} \\ \mathit{Just}\ (\mathit{Thread}\ t{\nearrow}\ \mathit{ss}{\nearrow})\end{array} = & \begin{array}{l}\mathit{Just}\ (\mathit{Thread}\ t{\nearrow}\ (\mathit{Signal} \rightarrow (c_0 \diamond c)\ a{\nearrow} : \mathit{ss}{\nearrow})) \\ \mathit{Nothing}\end{array} \\
f\ \_ & \begin{array}{l}\mathit{Nothing} \\ \mathit{Nothing}\end{array} = & \begin{array}{l}\mathit{Nothing} \\ \mathit{Nothing}\end{array}
\end{array}
\end{array}
$$

Furthermore, weaving actually processes not a just list of crossings but more generally a list of commands. Crossings are by far the most common kind of commands, but there are many other kinds.

An *advance* command shifts all future crossing locations by the specified amount, as if that much space has been consumed by the weaving. The amount to shift is the first state component in the M monad.

$$
\begin{array}{l}
\mathit{advance} :: \mathit{Monoid}\ c \Rightarrow c \rightarrow M\ c\ a\ t\ () \\
\mathit{advance}\ c = \mathit{modify}\ (\lambda(c_0, \mathit{threads}).\, (c_0 \diamond c, \mathit{threads}))
\end{array}
$$

A *through* command forces a thread to go through a location without crossing any other thread.

$$
\begin{array}{l}
\mathit{through} :: \mathit{Monoid}\ c \Rightarrow \mathit{Int} \rightarrow c \rightarrow a \rightarrow M\ c\ a\ t\ () \\
\mathit{through}\ i\ c\ a = \mathit{modify}\ (\lambda(c_0, \mathit{threads}).\, (c_0, \mathit{at1}\ i\ (\mathit{fmap}\ (f\ c_0))\ \mathit{threads}))\ \mathbf{where} \\
\quad f\ c_0\ (\mathit{Thread}\ t\ \mathit{ss}) = \mathit{Thread}\ t\ (\mathit{Signal} \rightarrow (c_0 \diamond c)\ a : \mathit{ss})
\end{array}
$$

A *begin* command turns an imaginary thread into a real one—in other words, puts the pen down. The second argument to *begin* (of type t) specifies information about the new real thread such as its identity and stroke color and whether its two ends should be connected to form a loop.

$$
\begin{array}{l}
\mathit{begin} :: \mathit{Eq}\ t \Rightarrow \mathit{Int} \rightarrow t \rightarrow M\ c\ a\ t\ () \\
\mathit{begin}\ i\ t = \mathit{modify}\ (\lambda(c_0, \mathit{threads}).\, (c_0, \mathit{at1}\ i\ f\ \mathit{threads}))\ \mathbf{where}
\end{array}
$$

$$
\begin{aligned}
&\quad f\ (\mathit{Just}\ \_) = \mathit{error}\ (\texttt{"Thread already begun at "} +\!\!+\ \mathit{show}\ i)\\
&\quad f\ \mathit{Nothing} = \mathit{Just}\ (\mathit{Thread}\ t\ [\,])
\end{aligned}
$$

Dually, an *end* command raises the pen, by moving a real thread to the end of the list and putting an imaginary thread where the real thread was.

$$
\begin{aligned}
&\mathit{end} :: \mathit{Eq}\ t \Rightarrow \mathit{Int} \to t \to M\ \ c\ \ a\ \ t\ ()\\
&\mathit{end}\ i\ t = \mathit{modify}\ (\lambda(c_0, \mathit{threads}).\,(c_0, f\ (\mathit{splitAt}\ i\ \mathit{threads})))\ \textbf{where}\\
&\quad f\ (\mathit{above}, \mathit{thread}@(\mathit{Just}\ (\mathit{Thread}\ t'\ \_)) : \mathit{below})\\
&\qquad \mid t \equiv t' \qquad\qquad = \mathit{above} +\!\!+\ \mathit{Nothing} : \mathit{below} +\!\!+\ [\mathit{thread}]\\
&\qquad \mid \mathit{otherwise} \quad\ \ = \mathit{error}\ (\texttt{"Different thread begun at "} +\!\!+\ \mathit{show}\ i)\\
&\quad f\ (\_, \mathit{Nothing} : \_) = \mathit{error}\ (\texttt{"Thread never begun at "} +\!\!+\ \mathit{show}\ i)\\
&\quad f\ (\_, [\,]) \qquad\qquad\ = \mathit{error}\ (\texttt{"Thread index "} +\!\!+\ \mathit{show}\ i +\!\!+\ \texttt{" out of range"})
\end{aligned}
$$

To carry out a command, we typically start with no shift and a list full of imaginary threads, and do not care about the final shift.

$$
\begin{aligned}
&\mathit{weave}' :: \mathit{Monoid}\ c \Rightarrow M\ \ c\ \ a\ \ t\ () \to (c, [\mathit{Maybe}\ (\mathit{Thread}\ \ c\ \ a\ \ t)])\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \to (c, [\mathit{Maybe}\ (\mathit{Thread}\ \ c\ \ a\ \ t)])\\
&\mathit{weave}' = \mathit{execState}\\
&\mathit{weave} :: \mathit{Monoid}\ c \Rightarrow M\ \ c\ \ a\ \ t\ () \to \mathit{Int} \to [\mathit{Maybe}\ (\mathit{Thread}\ \ c\ \ a\ \ t)]\\
&\mathit{weave}\ \mathit{cmd}\ \mathit{nThreads} = \mathit{snd}\ (\mathit{weave}'\ \mathit{cmd}\ (\mathit{mempty}, \mathit{replicate}\ \mathit{nThreads}\ \mathit{Nothing}))
\end{aligned}
$$

3. Space

Now we are ready to weave some threads in 2D. First we need some basic functions on 2D vectors.

$$
\begin{aligned}
&\textbf{type}\ \mathit{Coord}\ \ = \mathit{Double}\\
&\textbf{data}\ \mathit{Coords} = \langle \mathit{Coord}, \mathit{Coord}\rangle\\
&\textbf{infixl}\ 6 \oplus, \ominus\\
&\textbf{infixl}\ 7 \otimes, \oslash\\
&(\oplus), (\ominus) && :: \mathit{Coords} \to \mathit{Coords} \to \mathit{Coords}\\
&\langle x_1, y_1\rangle \oplus \langle x_2, y_2\rangle && = \langle x_1 + x_2, y_1 + y_2\rangle\\
&\langle x_1, y_1\rangle \ominus \langle x_2, y_2\rangle && = \langle x_1 - x_2, y_1 - y_2\rangle\\
&(\otimes), (\oslash) && :: \mathit{Coords} \to \mathit{Coord} \to \mathit{Coords}\\
&\langle x, y\rangle \otimes t && = \langle x \times t, y \times t\rangle\\
&\langle x, y\rangle \oslash t && = \langle x / t, y / t\rangle\\
&\ominus\cdot, \mathit{normalize} && :: \mathit{Coords} \to \mathit{Coords}\\
&\ominus\langle x, y\rangle && = \langle \mathit{negate}\ x, \mathit{negate}\ y\rangle\\
&\mathit{normalize}\ z && = z \oslash |z|\\
&|\cdot| && :: \mathit{Coords} \to \mathit{Coord}\\
&|\langle x, y\rangle| && = \sqrt{x \times x + y \times y}\\
&\textbf{instance}\ \mathit{Show}\ \mathit{Coords}\ \textbf{where}\\
&\quad \mathit{showsPrec}\ p\ \langle x, y\rangle = \mathit{showParen}\ (p > 10)\ (s\ x \circ \mathit{showChar}\ \texttt{','} \circ s\ y)\\
&\qquad \textbf{where}\ s \qquad\quad\ \ = \mathit{showFFloat}\ (\mathit{Just}\ 5)
\end{aligned}
$$

$$
\begin{array}{l}
\textbf{instance}\ \mathit{Monoid}\ \mathit{Coords}\ \textbf{where} \\
\quad \mathit{mempty} \qquad\qquad\quad = \langle 0, 0 \rangle \\
\quad \cdot \diamond \cdot \qquad\qquad\qquad = (\oplus)
\end{array}
$$

The following function *arg* takes two vectors as arguments and computes the cos and sin of the angle counterclockwise from the second vector to the first.

$$
\begin{array}{l}
\mathit{arg} :: \mathit{Coords} \to \mathit{Coords} \to \mathit{Coords} \\
\mathit{arg}\ \langle x_1, y_1 \rangle\ \langle x_2, y_2 \rangle = \mathit{normalize}\ \langle x_1 \times x_2 + y_1 \times y_2, y_1 \times x_2 - x_1 \times y_2 \rangle
\end{array}
$$

As promised, we record at each crossing the location and the tangent directions of the two threads. We record the pen color of each thread as a string. We also record whether the two ends of each thread should be connected to form a loop.

$$
\begin{array}{lll}
\textbf{type}\ \mathit{M\_}\ t & = M & \mathit{Coords}\ \mathit{Coords}\ t \\
\textbf{type}\ \mathit{Signal\_} & = \mathit{Signal} & \mathit{Coords}\ \mathit{Coords} \\
\textbf{type}\ \mathit{Thread\_} & = \mathit{Thread} & \mathit{Coords}\ \mathit{Coords}
\end{array}
$$

$$
\begin{array}{lll}
\textbf{data}\ \mathit{OC} & = \mathit{Open} \mid \mathit{Closed} & \textbf{deriving}\ (\mathit{Eq}, \mathit{Show}, \mathit{Read}) \\
\textbf{data}\ \mathit{Paint} & = \mathit{Paint}\ \mathit{String}\ \mathit{OC} & \textbf{deriving}\ (\mathit{Eq}, \mathit{Show}, \mathit{Read})
\end{array}
$$

$$
\begin{array}{l}
\mathit{paint}, \mathit{red}, \mathit{orange} :: \mathit{OC} \to \mathit{Paint} \\
\mathit{paint} \quad = \mathit{Paint}\ \texttt{""} \\
\mathit{red} \quad\ \ = \mathit{Paint}\ \texttt{"red"} \\
\mathit{orange} = \mathit{Paint}\ \texttt{"orange"}
\end{array}
$$

We draw a thread by alternating between cubic Bézier segments (between crossings) and straight line segments (at crossings; only if →, not if ⇢). The Bézier control points are chosen as by Hobby (1986). The length of the straight line segment is twice the length of the tangent direction vector specified. At each of the two ends of a thread is an additional line segment, whose length is *stub*.

$$
\begin{array}{l}
\mathit{stub} :: \mathit{Coord} \\
\mathit{stub} = 5
\end{array}
$$

$$
\begin{array}{l}
\mathit{hobby} :: \mathit{Coords} \to \mathit{Coords} \to \mathit{Coords} \to \mathit{Coords} \to (\mathit{Coords}, \mathit{Coords}) \\
\mathit{hobby}\ z_0\ z_1\ w_0\ w_1 = (z_0 \oplus w_0 \otimes (\rho / (\ \ 3 \times \tau_0) \times n / |w_0|), \\
\qquad\qquad\qquad\qquad\ \ z_1 \oplus w_1 \otimes (\sigma / (-3 \times \tau_1) \times n / |w_1|))\ \textbf{where} \\
\quad d \qquad\qquad\qquad = z_1 \ominus z_0 \\
\quad n \qquad\qquad\qquad = |d| \\
\quad \langle \mathit{cos}\theta, \mathit{sin}\theta \rangle \quad = \mathit{arg}\ w_0\ d \\
\quad \langle \mathit{cos}\phi, \mathit{sin}\phi \rangle \quad = \mathit{arg}\ d\ w_1 \\
\quad a \qquad\qquad\qquad = \sqrt{2} \\
\quad b \qquad\qquad\qquad = 1/16 \\
\quad c \qquad\qquad\qquad = (3 - \sqrt{5})/2 \\
\quad \alpha \qquad\qquad\qquad = a \times (\mathit{sin}\theta - b \times \mathit{sin}\phi) \times (\mathit{sin}\phi - b \times \mathit{sin}\theta) \times (\mathit{cos}\theta - \mathit{cos}\phi) \\
\quad \rho \qquad\qquad\qquad = (2 + \alpha)/(1 + (1 - c) \times \mathit{cos}\theta + c \times \mathit{cos}\phi) \\
\quad \sigma \qquad\qquad\qquad = (2 - \alpha)/(1 + (1 - c) \times \mathit{cos}\phi + c \times \mathit{cos}\theta) \\
\quad \tau_0 \qquad\qquad\qquad = 1 \\
\quad \tau_1 \qquad\qquad\qquad = 1
\end{array}
$$

It is handy to transform thread coordinates by a function from $\mathbb{R}^2$ to $\mathbb{R}^2$. To transform tangent directions along with crossing locations, we need the function to return its own partial derivatives.

$$
\begin{aligned}
&\mathit{transform} :: (\mathit{Coords} \to (\mathit{Coords}, \mathit{Coords}, \mathit{Coords})) \to [\mathit{Signal}_{\_}] \to [\mathit{Signal}_{\_}] \\
&\mathit{transform}\ f\ \mathit{ss} = [\ \mathit{Signal}\ \mathit{ou}\ \mathit{cc}\ (a' \otimes (|a|/|a'|)) \\
&\qquad\qquad\qquad\ \ |\ \mathit{Signal}\ \mathit{ou}\ c\ a@\langle \mathit{ax}, \mathit{ay}\rangle \leftarrow \mathit{ss}, \\
&\qquad\qquad\qquad\quad \textbf{let}\ (\mathit{cc}, c'x, c'y) = f\ c \\
&\qquad\qquad\qquad\qquad\quad a' \qquad\quad = c'x \otimes \mathit{ax} \oplus c'y \otimes \mathit{ay}]
\end{aligned}
$$

We can transform a picture on a cylinder into a picture on an annulus, to see the final result of a construction. To this end, the *circular* command executes a subcommand and transforms the result so that the threads go around a circle and their ends connect to each other according to their initial and final positions in the thread list. In other words, the threads are concatenated according to how their end points are identified along the seam of the cylinder. The circumference of the circle and the phase along the circle are determined by the total *advance* amount in the subcommand: the straight line segment between the origin and the total *advance* amount will be curled up into a circle while preserving the total length and the initial and final tangent direction.

$$
\begin{aligned}
&\mathit{circular} :: (\forall t'.M_{\_}\ t'\ ()) \to M_{\_}\ t\ () \\
&\mathit{circular}\ \mathit{cmd} = \mathit{modify}\ (\lambda(c_0, \mathit{threads}).\,(c_0, \mathit{zipWith}\ (g\ c_0)\ \mathit{threads}\ [0\,..])) \\
&\quad \textbf{where}\ (c_1, \mathit{threads1}) = \mathit{weave}'\ \mathit{cmd}\ (\mathit{mempty}, [\mathit{Just}\ (\mathit{Thread}\ i\ [\,])\ |\ i \leftarrow [0\,..]]) \\
&\qquad\quad g\ \_\ \mathit{Nothing}\ \_ = \mathit{Nothing} \\
&\qquad\quad g\ c_0\ (\mathit{Just}\ (\mathit{Thread}\ t\ \mathit{ss}))\ i = \mathit{Just}\ (\mathit{Thread}\ t\ (\mathit{transform}\ (f\ c_0)\ (\mathit{loop}\ i) +\!\!+\ \mathit{ss})) \\
&\qquad\quad \mathit{loop}\ i = \mathit{ss} +\!\!+\ \mathit{go}\ j \\
&\qquad\qquad \textbf{where}\ \mathit{Just}\ (\mathit{Thread}\ j\ \mathit{ss}) = \mathit{threads1}\ !!\ i \\
&\qquad\qquad\qquad\quad \mathit{go}\ j\ |\ i \equiv j \qquad = [\,] \\
&\qquad\qquad\qquad\qquad\quad\ |\ \mathit{otherwise} = \mathit{ss} +\!\!+\ \mathit{go}\ k \\
&\qquad\qquad\qquad\qquad \textbf{where}\ \mathit{Just}\ (\mathit{Thread}\ k\ \mathit{ss}) = \mathit{threads1}\ !!\ j \\
&\qquad\quad p \qquad\quad = |c_1| \\
&\qquad\quad \langle \mathit{rc}, \mathit{rs}\rangle \ \ = c_1 \oslash p \\
&\qquad\quad r_0 \qquad\ \ = p/(2 \times \pi) \\
&\qquad\quad f\ c_0\ \langle x, y\rangle = \textbf{let}\ \langle x', y'\rangle = \langle \mathit{rc}, -\mathit{rs}\rangle \otimes x \oplus \langle \mathit{rs}, \mathit{rc}\rangle \otimes y \\
&\qquad\qquad\qquad\qquad\quad t \qquad\ = x'/p \\
&\qquad\qquad\qquad\qquad\quad r \qquad\ = y' + r_0 \\
&\qquad\qquad\qquad\qquad\quad s \qquad\ = \sin\ (2 \times \pi \times t) \\
&\qquad\qquad\qquad\qquad\quad c \qquad\ = \cos\ (2 \times \pi \times t) \\
&\qquad\qquad\qquad\qquad\quad x'' \qquad = r \times s \\
&\qquad\qquad\qquad\qquad\quad y'' \qquad = r \times c - r_0 \\
&\qquad\qquad\qquad\qquad\quad d\ \langle \mathit{dx}, \mathit{dy}\rangle = \textbf{let}\ \langle \mathit{dx}', \mathit{dy}'\rangle = \langle \mathit{rc}, -\mathit{rs}\rangle \otimes \mathit{dx} \oplus \langle \mathit{rs}, \mathit{rc}\rangle \otimes \mathit{dy} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{dt} \qquad\quad = \mathit{dx}'/|c_1| \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{dr} \qquad\quad = \mathit{dy}' \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{ds} \qquad\quad = 2 \times \pi \times \mathit{dt} \times c \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{dc} \qquad\quad = -2 \times \pi \times \mathit{dt} \times s \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{dx}'' \qquad = \mathit{dr} \times s + r \times \mathit{ds}
\end{aligned}
$$

$$
\begin{aligned}
&\qquad\qquad\qquad\qquad\qquad\quad dy'' \qquad = dr \times c + r \times dc\\
&\qquad\qquad\qquad\quad \mathbf{in}\ \langle rc, rs\rangle \otimes dx'' \oplus \langle -rs, rc\rangle \otimes dy''\\
&\mathbf{in}\ (c_0 \oplus \langle rc, rs\rangle \otimes x'' \oplus \langle -rs, rc\rangle \otimes y'', d\ \langle 1, 0\rangle, d\ \langle 0, 1\rangle)
\end{aligned}
$$

We use rectangles to track the bounding box of a picture.

$$
\begin{aligned}
&\mathbf{data}\ \mathit{Rect} \quad\ = \mathit{Rect}\ \mathit{Coords}\ \mathit{Coords} \qquad \mathbf{deriving}\ \mathit{Show}\\
&\mathbf{data}\ \mathit{Bounds} = \mathit{Empty} \mid \mathit{Nonempty}\ \mathit{Rect}\ \mathbf{deriving}\ \mathit{Show}
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{instance}\ \mathit{Monoid}\ \mathit{Bounds}\ \mathbf{where}\\
&\quad \mathit{mempty} = \mathit{Empty}\\
&\quad \mathit{Empty} \diamond b = b\\
&\quad b \diamond \mathit{Empty} = b\\
&\quad \mathit{Nonempty}\ (\mathit{Rect}\ \langle x_1, y_1\rangle\ \langle x_2, y_2\rangle) \diamond \mathit{Nonempty}\ (\mathit{Rect}\ \langle x_1', y_1'\rangle\ \langle x_2', y_2'\rangle) =\\
&\qquad \mathit{Nonempty}\ (\mathit{Rect}\ \langle \mathit{min}\ x_1\ x_1', \mathit{min}\ y_1\ y_1'\rangle\ \langle \mathit{max}\ x_2\ x_2', \mathit{max}\ y_2\ y_2'\rangle)
\end{aligned}
$$

$$
\mathbf{class}\ \mathit{HasBounds}\ a\ \mathbf{where}\ \mathit{bounds} :: a \to \mathit{Bounds}
$$

$$
\begin{aligned}
&\mathbf{instance}\ \mathit{HasBounds}\ \mathit{Signal}_{\_}\ \mathbf{where}\\
&\quad \mathit{bounds}\ (\mathit{Signal}\ \_\ c\ a) = \mathit{Nonempty}\ (\mathit{Rect}\ (c \ominus d)\ (c \oplus d))\\
&\qquad \mathbf{where}\ \langle dx, dy\rangle = a \otimes (1 + \mathit{stub}/|a|)\\
&\qquad\qquad\quad d \qquad\ = \langle \mathit{abs}\ dx, \mathit{abs}\ dy\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{instance}\ \mathit{HasBounds}\ a \Rightarrow \mathit{HasBounds}\ [a]\ \mathbf{where}\\
&\quad \mathit{bounds} = \mathit{mconcat} \circ \mathit{map}\ \mathit{bounds}
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{instance}\ \mathit{HasBounds}\ (\mathit{Thread}_{\_}\ t)\ \mathbf{where}\\
&\quad \mathit{bounds}\ (\mathit{Thread}\ \_\ \mathit{signals}) = \mathit{bounds}\ \mathit{signals}
\end{aligned}
$$

$$
\begin{aligned}
&\mathbf{instance}\ \mathit{HasBounds}\ a \Rightarrow \mathit{HasBounds}\ (\mathit{Maybe}\ a)\ \mathbf{where}\\
&\quad \mathit{bounds} = \mathit{maybe}\ \mathit{Empty}\ \mathit{bounds}
\end{aligned}
$$

4. SVG output

The function *path* draws a thread history as an SVG path. If the first argument is *Closed*, then *path* draws a closed thread (which requires a closed SVG path if and only if there is no gap ($\dashrightarrow$) in the thread).

$$
\begin{aligned}
&\mathit{path} :: \mathit{OC} \to [\mathit{Signal}_{\_}] \to \mathit{String}\\
&\mathit{path}\ \mathit{Open}\ [\,] = \texttt{""}\\
&\mathit{path}\ \mathit{Open}\ (\mathit{Signal}\ \mathit{ou}\ c\ a : \mathit{ss}) =\\
&\quad \texttt{'M'} : \mathit{show}\ (c \oplus a \otimes (1 + \mathit{stub}/|a|)) +\!\!+\\
&\quad \mathbf{case}\ \mathit{ou}\ \mathbf{of} \rightarrow\ \to \mathit{path}' \rightarrow c\ a\ \mathit{ss}\ \mathit{True}\\
&\qquad\qquad\qquad \dashrightarrow\ \to \texttt{'L'} : \mathit{show}\ (c \oplus a) +\!\!+ \mathit{path}' \dashrightarrow c\ a\ \mathit{ss}\ \mathit{True}\\
&\mathit{path}\ \mathit{Closed}\ \mathit{ss} =\\
&\quad \mathbf{case}\ \mathit{span}\ (\lambda(\mathit{Signal}\ \mathit{ou}\ \_\ \_).\ \mathit{ou} \equiv \rightarrow)\ \mathit{ss}\ \mathbf{of}\\
&\qquad ([\,], [\,]) \to \texttt{""}\\
&\qquad (s@(\mathit{Signal} \rightarrow c\ a) : \mathit{rest}, [\,]) \to \mathit{path}' \dashrightarrow c\ a\ (\mathit{rest} +\!\!+ [s])\ \mathit{False} +\!\!+ [\texttt{'Z'}]\\
&\qquad (\mathit{initial}, s@(\mathit{Signal} \dashrightarrow c\ a) : \mathit{rest}) \to \mathit{path}' \dashrightarrow c\ a\ (\mathit{rest} +\!\!+ \mathit{initial} +\!\!+ [s])\ \mathit{False}\\
&\qquad \_ \to \mathit{error}\ \texttt{"Internal error: unexpected pattern-match failure"}
\end{aligned}
$$

$$
\mathit{path}' :: \mathit{OU} \to \mathit{Coords} \to \mathit{Coords} \to [\mathit{Signal}_{\_}] \to \mathit{Bool} \to \mathit{String}
$$

```
path' _ _ _ [] False = ""
path' ou c a [] True =
  (case ou of ➝ → ""; ⇢ → 'M') : show (c ⊖ a)) ++
  'L' : show (c ⊖ a ⊗ (1 + stub/|a|))
path' ou₁ c₁ a₁ (Signal ou₂ c₂ a₂ : ss) finalStub =
  (case ou₁ of ➝ → 'L'; ⇢ → 'M') : show (c₁ ⊖ a₁) ++
  'C' : show b₁ ++ ' ' : show b₂ ++ ' ' : show (c₂ ⊕ a₂) ++
  path' ou₂ c₂ a₂ ss finalStub
  where (b₁, b₂) = hobby (c₁ ⊖ a₁) (c₂ ⊕ a₂) (⊖a₁) (⊖a₂)
```

The following function draws a bunch of threads as an SVG document.

```
svg :: [Maybe (Thread_ Paint)] → String
svg threads = unlines
   $ ("<svg xmlns=\"http://www.w3.org/2000/svg\" width=\"" ++
              show width ++ "\" height=\"" ++ show height ++ "\">")
   : ("<g transform=\"scale(5 -5) translate(" ++
              show tx ++ " " ++ show ty ++ ")\">")
   : [ "<path fill=\"none\" stroke=\"" ++ stroke ++
              "\" stroke-width=\".8\" d=\"" ++ path oc ss ++ "\"/>"
     | Just (Thread (Paint paint oc) ss) ← threads,
       let stroke = if null paint then "currentColor" else paint]
   ++ ["</g></svg>"]
  where scale  = 5
        margin = 2
        (⟨tx, ty⟩, ⟨width, height⟩) =
          case bounds threads of
            Empty → (⟨0, 0⟩, ⟨0, 0⟩)
            Nonempty (Rect ⟨x₁, y₁⟩ ⟨x₂, y₂⟩) →
              (⟨margin − x₁, −margin − y₂⟩,
               ⟨x₂ − x₁ + 2 × margin, y₂ − y₁ + 2 × margin⟩ ⊗ scale)
```

We can also put the same picture in TikZ.

```
tikz :: [Maybe (Thread_ Paint)] → String
tikz threads = unlines
   $ "\\begin{tikzpicture}[line width=.8pt]"
   : [ "\\draw " ++ options ++ "svg \"" ++ path oc ss ++ "\";"
     | Just (Thread (Paint paint oc) ss) ← threads,
       let options = if null paint then "" else "[draw=" ++ paint ++ "] "]
   ++ ["\\end{tikzpicture}%"]
```

References

Hobby, John D. 1986. Smooth, easy to computer interpolating splines. *Discrete and Computational Geometry* 1(1):123–140.