Mandatory assignment

In this assignment you will make a program in SML and write a technical report documenting the program.

Contents of the report

1. *Front page*, containing title, author(s), date, course, and institution.
2. *Table of contents.*
3. *Introduction* (max. 1 page), stating the aim of the report, the required prerequisites of the reader, plus a short summary of the remaining sections of the report.
4. *User's guide* (max. 4 pages).
5. *Examples* (max. 4 pages) on the use of the program.
6. *Technical documentation* of the program.
7. *Conclusion* (max. 1 page), assessing the state of the solution and what could be improved.

The form of the report

The report should be *brief* and *clear* (the stated number of pages is the maximum limit). The *contents* of each section in the report should be clear and *limited*, and you should stick to it, i.e.: Make clear to yourself what the reader should expect to find in the section, and write about that, and *only that*. Each section or subsection should have a uniform style, either as a *tutorial* where the reader can get an overview of a subject, or as a *reference* where the reader can look up specific information (with a minimum of reading). Do not repeat the same information – use (backward) references instead.

The report should be written for a reader who has no knowledge of the specific problem to be solved, but who has otherwise the same general knowledge as yourself. The report should be of such a quality that you would accept it as good program documentation.

The report should be made by use of electronic text processing in order to ease the corrections and changes which will be part of the writing process.

Problem

You will make a library of SML functions for manipulation and drawing of piecewise linear curves. Such a curve is drawn as a sequence of segments $P_1P_2,\ P_2P_3,\ \ldots,\ P_{n-1}P_n$ where $P_1, P_2, \ldots, P_n$ are points in the plane. The point P_1 is called the *start point* of the curve, and the point P_n is called the *end point* of the curve. We use usual rectangular, Cartesian coordinates in the plane, so points and vectors in the plane are represented by coordinates which are pairs of real numbers. The point with coordinates `(0.0,0.0)` is called the *origin* of the plane.

You will make the following functions on curves:

point Computes the curve with start point and end point equal to a given point.

line Computes the curve P_1P_2 from the points P_1 and P_2.

join For given curves c_1 and c_2, this function computes the curve c consisting of three parts: (1) the curve c_1, (2) the segment from the end point of c_1 to the start point of c_2, and (3) the curve c_2.

trans For given curve c and vector $\mathbf{v}$, this function computes the curve obtained from c under the parallel translation in the plane determined by the vector $\mathbf{v}$.

rot For given curve c and angle t, this function computes the curve c' obtained from c under the rotation of angle t around the origin of the coordinate system.

reflect1 For given curve c, this function computes the curve c' obtained from c by reflection in the first axis of the coordinate system.

reflect2 For given curve c, this function computes the curve c' obtained from c by reflection in second axis of the coordinate system.

scale For given curve c and real number c, this function computes the curve c' obtained from c by multiplication by factor c from the origin of the coordinate system.

size1 Horizontal size of the curve, i.e. the difference between the largest and smallest first coordinates for points of the curve.

size2 Vertical size of the curve, i.e. the difference between the largest and smallest second coordinates for points of the curve.

draw Outputs the curve to a file on the disk in *PostScript* format (cf. the remarks below).

Points and vectors in the plane. Rotation and multiplication

Points and vectors in the plane are given by their coordinates which are pairs of real numbers.

The *translation* determined by a vector $\mathbf{v} = (v_1, v_2)$ maps a point P with coordinates (x, y) into the point P' with coordinates (x', y') where:

$$\begin{aligned} x' &= x + v_1 \\ y' &= y + v_2 \end{aligned}$$

The rotation of angle t (where t is a real number) around the origin maps a point P with coordinates (x, y) into the point P' with coordinates (x', y') where:

$$\begin{aligned} x' &= x \cos t - y \sin t \\ y' &= x \sin t + y \cos t \end{aligned}$$

Note that the library functions `Math.sin` and `Math.cos` use the angle expressed in radians. You may use the value `Math.pi` for the mathematical constant π when converting between radians and degrees.

The *reflection* in the first axis of the coordinate system maps a point P with coordinates (x, y) into the point P' with coordinates $(x', y') = (x, -y)$.

The *reflection* in the second axis of the coordinate system maps a point P with coordinates (x, y) into the point P' with coordinates $(x', y') = (-x, y)$.

The *multiplication* by factor c from the origin of the coordinate system maps a point P with coordinates (x, y) into the point P' with coordinates $(x', y') = (c\,x, c\,y)$.

Output of a curve. PostScript form

The language *PostScript* is used for representing printed pages (containing graphics). The curve consisting of the segments joining the points (100.0, 100.0), (200.0, 100.0) and (150.0, 186.60) may e.g. be represented by the PostScript instructions below.

Apart from the intial and final conjurations (to please the PostScript interpreter) there has to be a PostScript `moveto` command containing the coordinates of the start point, a PostScript `lineto` commands containing the coordinates of the end point followed by a PostScript command `stroke` for each segment in the curve.

Note that the coordinates are written in decimal form. The conversion to this form can be made using the `Real.fmt` function, but negative values must use the minus sign '`-`' (and not '`~`').

```
%!
1 1 scale
newpath

100.0 100.0 moveto
200.0 100.0 lineto
stroke

200.0 100.0 moveto
150.0 186.60 lineto
stroke

showpage
```

When programming the function **draw** it is convenient to use an auxiliary function building the string which represents the curve in PostScript form. One may then use the function `wrstring` below to output the string (= the first component of the argument) to a file on the disk (with name = the second component of the argument):

```
fun wrstring(data,filename) =
   let val out = TextOI.openOout(filename) in
       TextOI.output(out,data);
       TextOI.closeOut(out)
   end
```

NB: Note that this function will overwrite an existing file "*filename*".

A file in PostScript format can be output directly on the laser printer in the data bar as the printer will interpret the commands and make the corresponding drawing. The point with coordinates (0,0) corresponds to the lower left corner of the sheet while 72 units correspond to one US inch.

A file in PostScript format may also be shown on the screen using the "`ghostview`" program in an `xterm`:

`ghostview` *filename* `&`

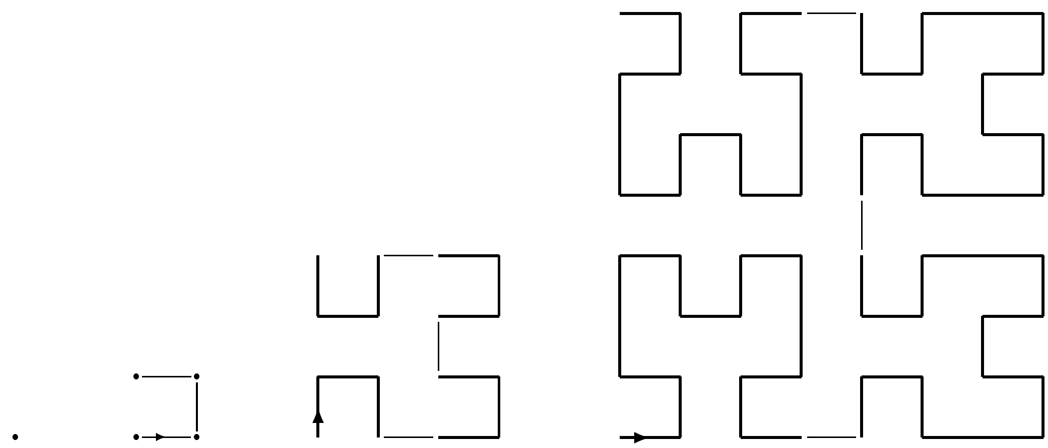
Applications

The functions should be used to program functions for Hilbert curves, Peano curves and Sierpinski curves – as described in the following.

When drawing these curves using the function **draw**, one should first obtain a suitable scale and placement of the curve by using the functions **scale** and **trans**.

Hilbert curves

The Hilbert curves $h_0, h_1, h_2, \ldots$ are a system of curves, where the curve h_{n+1} is formed by connecting 4 curves $h_{n1}, h_{n2}, h_{n3}, h_{n4}$ which are obtained from the curve h_n by transformations composed of reflections, rotations and translations. The figure shows the Hilbert curves h_0, h_1, h_2 and h_3 and how the curves h_1, h_2 and h_3 are composed of four parts:



h_0 h_1 h_2 h_3

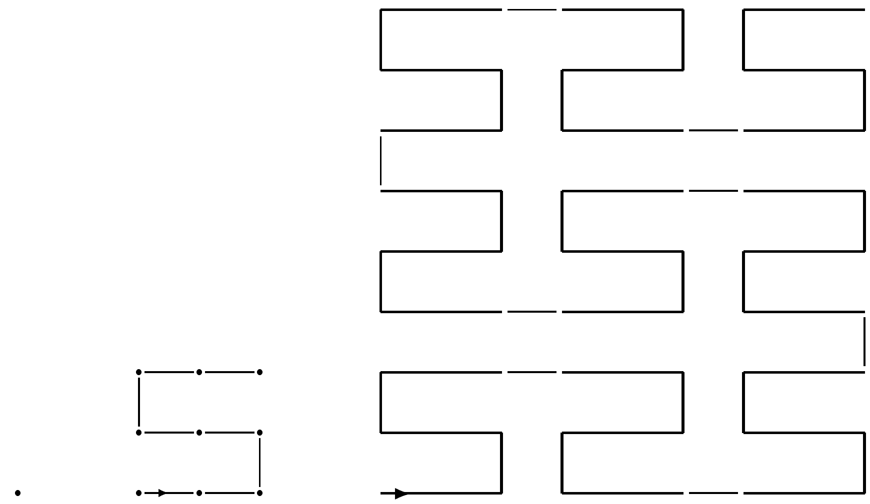
All curves start in the origin and the connecting segments (thin lines in the Figure) are of length 1.

Declare the function **hilbert** which computes the curve h_{n+1} from the curve h_n for any n. Use this function to make a program drawing the curve h_4 (in a suitable scale).

Peano curves

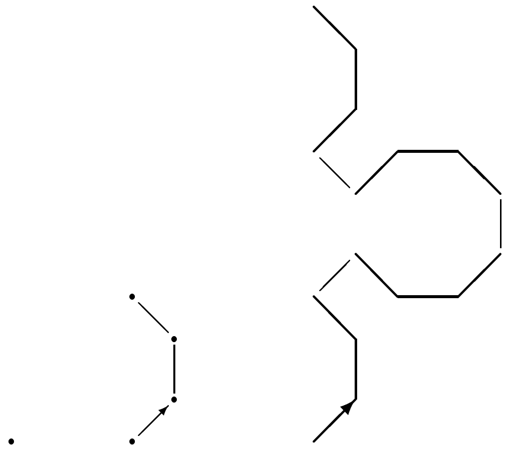
The Peano curves $p_0, p_1, p_2, \ldots$ are a system of curves, where the curve p_{n+1} is obtained by connecting 9 curves $p_{n1}, p_{n2}, \ldots, p_{n9}$ which are obtained from the curve p_n by transformations composed of reflections, rotations and translations. The figure shows the Peano curves p_0, p_1 og p_2 and how the curves p_1 and p_2 are composed of 9 parts. All curves start in the origin and the connecting segments (thin lines in the Figure) are of length 1.

Declare the function **peano** which computes the curve p_{n+1} from the curve p_n for any n. Use this function to make a program drawing the curve p_3 (in a suitable scale). (It may be convenient to collect the 9 curves p_{n1},..., p_{n9} in 3 groups each consisting of 3 curves.)



p_0 p_1 p_2

Sierpinski curves



s_0 s_1 s_2

The Sierpinski curves $s_0, s_1, s_2, \ldots$ are a system of curves, where the curve s_{n+1} is obtained by connecting four curves $s_{n1}, s_{n2}, s_{n3}, s_{n4}$ which are obtained from the curve s_n by transformations composed of reflections, rotations and translations. The figure shows the Sierpinski curves s_0, s_1 and s_2 and how the curves s_1 and s_2 are composed of four parts. Note that all segments in a Sierpinski curve have same length. All curves start in the origin and the connecting segments (thin lines in the Figure) are of length 1.

Declare the function **sierpinski** which computes the curve s_{n+1} from the curve s_n for any n, and use this function to make a program drawing the curve s_4 (in a suitable scale).