Comp3402 Advanced Lindenmayer Systems

C.B.Price December 2022

Purpose	(i) To apply your understanding of L-Systems to more advanced situations.
Files Required	Octave
ILO Contribution	LO 4
Send to Me	nix
Homework	Read chapter 12 (WIP)

I don't expect you to do all these (or indeed any) but have a go at anything that you find interesting

---

1. **Sequences of Shapes.** This activity is about how to construct sequences or chains of shapes. Consider the following grammar

grammar	Axiom	X
	Rule	X→ AX

where A is a string made up of the usual symbols F, +, -, e.g., F+

Now the strings generated by this grammar are

depth	**str**
0	X
1	AX
2	AAX
3	AAAX
4	AAAX

so you can see the string A is concatenated, added to the end of the previous string. The turtle will draw this concatenated shape *provided the shape has different entry and exit points*. An example of a shape which violates this condition is a square.

(a) Code up the grammar with A = F+ and set the angle to 30 degs. Now explore the result

(b) Create your own interesting shape and its string A and use the above grammar to experiment.

---

2. **Concatenation Grammars.** If you are a theoretician, you might like to try these alternative 'concatenation' grammars and see if there are any differences

grammar A	Axiom	AX
	Rule	X→ AX

grammar B	Axiom	AX
	Rule	X→ BX

grammar C	Axiom	AX
	Rule	X→ BAX

---

3. **Computation with L-Systems**. Let's investigate how L-Systems can generate numbers. While this is not exactly 'computation' it is quite close. Some questions have **hints** which can be found at the end of this document,

(a) **[Hint available]** Write a grammar (axiom and rule(s)) to generate odd numbers. So you should end up with this for successive depths

1
111
11111
1111111

You are allowed to have the symbol X anywhere.

(b) Now write a grammar to generate the even numbers, this is quite similar to (a)

11
1111
111111
11111111

(c) Now write a grammar to generate numbers where the next number is double the previous number. This is how binary numbers are arranged to encode denary numbers (1,2,3,…).

1	1
11	2
1111	4
11111111	8

(d) Repeat the above where the next number is 3 times the previous numbers like this.

1	1
111	3
111111111	9
111111111111111111111111111	27

(e) You may have noticed that (c) produces numbers given by this expression $n = 2^d$ where *d* is the depth, and that (d) produces numbers $n = 3^d$ where *d* is the depth. How would you go about creating a grammar to produce numbers $n = N^d$ where *N* could be 4, 5, 6, …?

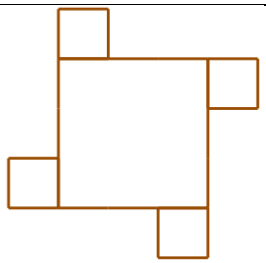
---

4. **Dr C's Conjecture.** A 'conjecture' is an idea which has not been proven, but the one who makes the conjecture believes that it could be proven. Here's my conjecture.

The following grammar will make a shape with corners, and at each corner it will draw a shape given by the string A*, where* A *can be any string accepted by the system*.

grammar	Axiom	not specified (except it must contain F)
	Rule	F→ F[A]+F[A]+F[A]+ …

Here's something I have in mind.



So what should you do?

(a) Choose a simple axiom (e.g. F) and complete the rule by specifying a string for A. What do you get for various depths?

(b) Now thinking time. Can you *generalize* the above rule using something more complex than an F and a +?

---

5. **Deep Theoretical Question. Do not read any further.** The L-System I have created cannot do any of the following: (i) adding or subtracting two numbers, (ii) multiplying or dividing two numbers, (iii) testing if one number is larger than another. Without these, it cannot replace procedural programming.

The question is why? There is a *fundamental limitation* in the L-System I have created which prevents all of the above. Can you see what it is? If you can, and you think you can extend the code to deal with this, then that would make a great mini-project for Position Paper 3 part 1.

---

6. **Mission Impossible: The Cantor Set.** This is a set of lines. Each line corresponds to a depth of our L-System. Here's the set of lines for depths 0 (= the Axiom) to depth 3. You can see that each line is split into three lines (with the middle line missing … hint not drawn).

Each production run will generate just one of these lines, corresponding to the depth you input.

Also, there is an additional symbol in my L-System **f**. This works just like **F** except the turtle does not draw a line.

Your mission, should you choose to accept it, is to produce a grammar to compute the Cantor Set. Here's a starting point. The Axiom is obvious, it draws a line to the right.

grammar	Axiom	+F
	Rule1	F→ ??
	Rule2	f→ ??

---

7. **Producing a Line of Shapes.** Let's use the additional symbol **f** to produce a line of squares like this.

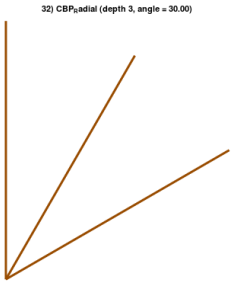
Let's recap what we know:

(i) We know the string to draw a square.
(ii) We know how to create a rule to concatenate things.
(iii) The new symbol **f** moves the turtle in its current direction without drawing a line.

(a) Have a go at reproducing the above. Try out higher depths.
(b) You may want to assemble a line of shapes you have created.
(c) Or you may want to work out how to produce a 2D grid of shapes.

---

8. **Radial Patterns.** Here we shall investigate how to produce a pattern of radial lines. Here's what I have in mind for my grammar with depth=3 and angle=30



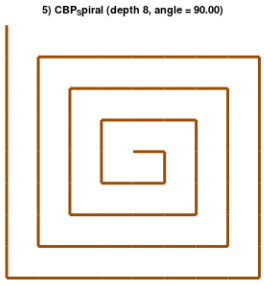
Note that all lines start at the same place. How to do this? Well, you know how to get the turtle to remember its position (and angle) by using the *push* symbol **[** then doing something, then returning home using the *pop* symbol **]**. So, the above diagram will need this sequence **[F]+**.

(a) Create the grammar using this sequence. It is also a concatenation, so you will need to use the concatenation pattern.

(b) Now create a grammar to produce a radial arrangement of patterns of your own choice.

---

9. **Spiral.** Create a grammar to produce this spiral pattern.



Hints 3(a) Here's a starting point.

grammar	Axiom	X
	Rule	X→