Symbol and Production implementation

The idea is to be able to apply a parametric L-system, a trivial example being something like

ω: A(1)

P_0: A(s) → F(s) A(s/r)

where r is an externally defined constant.

The way I dealt with this is something like the following.

- Symbols (e.g. A, F) are essentially user-defined objects, so the user is responsible (for the moment) for constructing a symbol that does what they want.

- User constructed symbols are, of course, restricted, specifically to implement the Symbol trait, which gives any symbol a set of known functions (i.e., an interface).

- The Symbol traits are representation, evaluate, and produce.

- representation: How the symbol is written down in an L-string.

- produce(constants): The set of new Symbol instances this Symbol expands into as an L-string containing it is read/interpreted, based on

  a) This symbols internal state (its parameters), Σ

  b) A set of externally defined constants, C

So produce is a function that takes in itself and a set of externally defined constants (C) and produces a new set of Symbol structs with their own, new, internal state

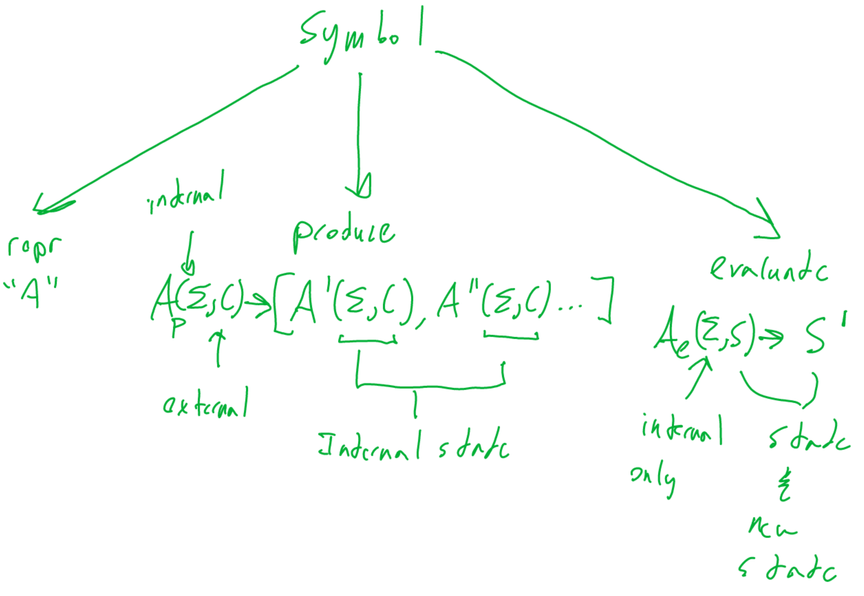
definite symbols

fn produce ($self, $C) => [$S_0, $S_1, ...]

- evaluate: How the symbol acts on a state (e.g. move the turtle forward, turn left, etc.) based only on its internal parameters Σ.

So it is a function that takes itself, and a state, and uses its internal state (parameters) to modify the incoming state to produce a new state.

fn evaluate ($self, $state) => state



For example, let's evaluate the system

ω: $A(1)$

p_0: $A(s) \Rightarrow F(s)\,A(s)$

p_1: $F(s) \Rightarrow F(s)$ (Implicit)

The first thing to do is to construct the A & F symbols

Let $A = A([s_A])$ where $s_A \in \Sigma_A$

and $F = F([s_F])$ where $s_F \in \Sigma_F$

such that A and F are both not much more than wrappers around a vector expected to contain the single parameter "s".

Next, we need to implement the symbol traits for A and F

A.representation(sc|F) = "$A([s_A])$"

A.produce(sc|F, C) = $F([s_A])\,A([{}^{s_A}/_{R}])$

(Roughly, A.produce $\sim A(s)$) — $C[0]$

A.evaluate(sc|F, S) = "Move S forward by s_A"

(This is just an example)

F.representation(sc|F) = "$F([s_F])$"

F.produce(sc|F, C) = $F([s_F])$

(C is ignored, here)

(Again, roughly F.produce $\sim F(s)$)

F.evaluate(sc|F, S) = "Move S forward by s_F"

(A and F apply the same instruction to S).

It's important to note here that when we write e.g. $A([s])$, we are constructing a new A symbol with internal state $s_A = s$.

So now we may interpret/evaluate the system

$\omega : A(1)$

$\rho_0 : A(s) \to F(s)\, A(s/R)$

$\rho_1 : F(s) \to F(s)$

define: $C = [(R = 1.456)]$

We start at ω, and read $A(1)$ as $A([s_A = 1])$, i.e. construct the symbol A for internal state $s_A = 1$.

We then produce $A([s_A = 1])$, to get

$A([s=1]).\text{produce}(C) = F([s=1])\, A([^{s=1}/_{R=1.456}])$

$= F([1])\, A([^{1}/_{1.456}])$

which we may produce again, of course, as follows (s_F)

$F([1]).\text{produce}(C) = F([s=1]) = F([1])$

$A([^{1}/_{1.456}]).\text{produce}(C) = F([s = \underbrace{\tfrac{1}{1.456}}_{s_A}])\, A([^{\overbrace{1/1.456}^{s_A}}/_{\underbrace{1.456}_{R}}])$

$= F([\tfrac{1}{1.456}])\, A([^{1/1.456}/_{1.456}])$

The resultant L-string is the linear combination of these productions, so

$F([1])\, A([\tfrac{1}{1.456}]) = F([1])\, F([\tfrac{1}{1.456}])\, A([^{1/1.456}/_{1.456}])$

which is what we want/expect.

Now, how could we improve this?

- Named parameters? (Internal state, constants)
- Optional constants? (e.g. not ignoring them in F_1 but not passing them on at all)
- ?