CA-DSL Initial Planning Document

Purpose and Concepts

Our programming language is designed to ease the ability to describe and visualize cellular automata. There are currently a variety of different format strings for certain specific classes of cellular automata and some visualizers that can visualize them, but we want a general approach that can enable for the construction of new classes of cellular automata in a simple, but systematic manner.

A *cellular automata* is a collection of *cells* which each have a *state*. At each time step, all the cells transition between states according to *rules*. Rules define the start and end states of the transition and the condition when the transition should occur. The condition is dependent on the state of cells in the *neighborhood* of a cell. In *totalistic cellular automata*, it is only the number of cells in each state which the condition depends on. In *non-totalistic cellular automata*, the positioning of the cell in each state is also factored into the condition. We want to make expressing totalistic cellular automata simple, while still making it possible to express non-totalistic cellular automata.

The concept of a neighborhood requires the introduction of a *topology*. A topology represents how cells are connected. In the canonical cellular automata, Conway's Game of Life, the topology is a flat 2D infinite plane. We can't model an infinite plane with a finite computer, so we intend to have the ability to specify certain cells which are frozen in one state, thus creating a wall around a finite portion of the plane. Our DSL will also enable modeling of topologies composed of hexagonal cells as well as looped topologies. Our dream is that we can model a cellular automata on a soccer ball.

Example Programs

Conway's Game of Life

```
(define conways
    (rule
        [(0 -> 1) (3 in 1)]
        [(1 -> 1) ((2 3) in 1)]))

(define world
    (make-world 100 100 (random 50 50)))

(run conways world)
```

Wireworld

```
(define wireworld ;; https://conwaylife.com/wiki/OCA:WireWorld
    (rule #default identity
        [('head -> 'tail)]
        [('tail -> 'conductor)]
        [('conductor -> 'head) ((1 2) in 'head)]))
```

Star Wars

```
(define star-wars ;; https://conwaylife.com/wiki/OCA:Star_Wars
    (rule #default add1
        [(0 -> 0) (not 2 in 1)]
        [(1 -> 1) ((3 4 5) in 1)]
        [(3 -> 0)]))
```

Grammars and Signatures

```
<RULE> := (rule <NEIGHBORHOOD> <DEFAULT> <BRANCH> ... )

<NEIGHBORHOOD> :=
   | #:neighborhood <INLINE-NEIGHBORHOOD>

<DEFAULT> :=
           | #:default <expr>

<BRANCH> := [<TRANSITION> <COND>]
        | [<TRANSITION>]
        | [<STATE> <expr>]



<TRANSITION> := (<STATE> -> <STATE>)

<COND> := <expr>
        | <exc?> <COUNTS> in <STATE>
        | <EXC?> <COUNTS> from <INLINE-NEIGHBORHOOD> are <STATE>
        | <EXC?> <CELL> is <STATE>

<COUNTS> := (<natural> <natural> ...)
         | <natural>

<EXC?> :=
        | except

<STATE> := <expr>
<CELL> := <expr>
<INLINE-NEIGHBORHOOD> := <list>
```

Milestones

- Create a `run` macro which can take render a cellular automata with no rules, just all cells staying in the same state.
- Create the `rule` macro, but without conditional checks, so all cells just loop through states
- Implement the conditional portion of the `rule` macro to expand out the < cond> grammar

- Augment the `from` clause of the cond grammar to support cells as well as offsets by prefixing cells with the keyword `absolute`.
- Add library functions to ease the creation of topologies.
- Add library functions to translate from existing cellular automata specification strings to our DSL