Cellular Automata Language

(working name)

Ramses Driskell (rd2491) - Position TBD (Possible Systems Arch.)
Percee Goings (pbg2111) - Position TBD (Possible Group Manager)
Fei-Tzin Lee (fl2301) - Position TBD (Possible Systems Integrator)
Geoffrey Loss (gml2151) - Tester and Validator
Andrei Tapai (amt2197) - Language Guru

Programming Language & Translators (COMS W4115)
Professor: Alfred Aho
Spring 2015
Tuesday, February 24, 2015

1.**What class of problems is your language intended to solve?**

Our cellular automata language (henceforth CAL) is intended for ease of expression of complex cellular automata. This encompasses everything from expressing 2D cell structure (square cells, hexagonal cells), initial state, finite number of possible states, and algorithms for updating the state of cells (both synchronously and asynchronously).

2. **Why do you think this class of problems is important/interesting?**

We think this class of problems is important because of the numerous applications cellular automata have both in computing and in understanding aspects of our physical world (such as biology). In addition, cellular automata have been proposed as possible functions for cryptographic purposes. Since at least two of our team members are interested in the security aspect of computing, cellular automata provide a new avenue of exploration and learning.Additionally, we find this class of problems interesting because watching simple deterministic rules produce complex behavior is honestly thought-provoking with respect to the rules governing our own world (and it looks cool).

3. **Are there existing languages aimed at this class of problems? If so, how is your language different/better?**

The inspiration for CAL was our Language Guru's (Andrei Tapai) first computer science course in high-school which introduced concepts through an application/language called NetLogo. While in NetLogo you could describe cellular automata, you are limited by the built-in GUI and by the necessity to perform limited GUI maintenance/ draw requests. CAL attempts to simultaneously simplify and expand the options available to the programmer. It does this by eliminating the need for the programmer to spend dozens of lines coding in visual aspects of the final executable. Additionally, NetLogo as a language is more catered toward developing multi-agent systems of varying complexity such as the Wolf-Sheep Predation Model and the Termites model. As such, its core objective is not development of 2D cellular automata.

Outside of NetLogo, cellular automata have been simulated in more mainstream languages such as C and Java. However, the recurring problem with this approach is the need to use often clunky visual libraries to develop a visual interpretation of your procedures. Needless to say, this is often a headache.

4. **Who are the intended users of your language? What background do you assume they will have?**

The intended users of CAL are those with some understanding of imperative languages such as C in addition to basic understanding of the workings of automata. Users should be familiar with variables and their initialization, writing functions that can depend on each other (but don't have to), and with computing concepts such as *state*. CAL is intended to be easy to understand to those who have already been introduced to the field of computer science.

5. **What properties would you like your language to have?**

1. Comments:

Comment:"This is a properly formatted comment"

2.Built-in types:

```
int a = 1
float b = 1.09
boolean c = true
string d = "Hello" (unsure if the language needs strings as types)

grid g = 2 * 2
```

3. Mathematical Operators:

```
a + b          (Addition)
b - c          (Subtraction)
c > d          (Greater Than)
d < e          (Less Than)
e <= f (Less Than or Equal To)
f >= g (Greater Than or Equal To)
g = h          (Equal To)
g != i (Not Equal To)
```

4. Control Flow

Control flow will function through the use of:

```
if ||
else-if||
else ||
```

etc.

5. Functions:

“return-type” “function-name” (function inputs)
|
(indented statements)
|

```
int add_increment (int a, int b)
|
       int c = 0
       c = a + b
       c ++
       return c
|
```

6. Keywords:

foreach neighbor that is “condition goes here”

cells track “properties go here”

```
g is hexagonal
```

6. **Give two or three simple prototypical concrete problems you'd like your language to solve. If you can sketch what programs might look like in your language to solve these problems, so much the better.**

One prototypical program would be programming Conway's Game of Life. The following is CAL's source for it. Note: This is our initial conception for our language's syntax, keywords, etc. Most of it is likely if not certain to go through multiple revisions.

```
Comment:"Conway's Game of Life plays out on an infinite two-dimensional grid of
square cells theoretically. Here we will limit ourselves to one hundred by one
hundred"

boolean alive
grid g = 100 * 100
g is square

cells have alive
cells track live-neighbors

int live-neighbors (cell c)
|
	int live = 0

	foreach neighbor that is (neighbor-of c)
	|
		if (neighbor is alive)
		|
			live++
		|
	|
|


void main ()
|
	foreach cell
	|
		if (alive && live-neighbors < 2)
		|

			alive = false
		|
		else-if (alive && live-neighbors > 3)
		|
```

```
                        alive = false
                |
                else (!alive && live-neighbors = 3)
                |
                        alive = true
                |
        |
|
```

We could provide code for other rulesets but the above snippet should serve as a general guide for how this language will pan out in the long term. In general, “problems” are solved by indicating an initial state and by functions describing a ruleset.