Eugene Kim
Calvin Hu
Nathan Keane

COMS4115 Proposal: Cellular Automata Language (CAL)

Motivation

Cellular automata are discrete, abstract computational systems that provide useful models of non-linear dynamics in various scientific fields. Cellular automata are composed of discrete, homogenous units called cells and each cell possesses one of a finite number of states, which changes at discrete time steps simultaneously with the states of the other cells. State change of a cell is governed by the states of its immediately surrounding cells. Cellular automata can display complex, evolving behavior of small, homogenous units following set rules and are used to study computation, algorithmic problems, pattern formation, abstract complexity science and theoretical biology.

Objective and Use

Our language, Cellular Automata Language (CAL), is intended for programmers to quickly and easily design cellular automata suited for their use. It should be easy for the programmers to designate the set of initial states and set rules associated with their own cellular automata. They should be able to see the outcome after a specific number of steps in both textual and graphical formats.

Description of Language

CAL will make use of a limited number of primitives to allow easy instantiation of rules and states. State of an entire cellular automaton will be encapsulated in a primitive called Grid. Initial state of cellular automata will be set either randomly or via input files. CAL will allow programmers to declare a rule succinctly by using our language syntax. Rules are also a primitive, along with Grid, Integer, and Boolean.

Language Syntax

All functions in CAL are pass-by-reference.
Double slash - "//" - will make all text to the right on the same line a comment.
A built in function - "void run(grid g, rule r)" - will do one iteration of rule r over grid g.
A built in function - "void display(grid g)" - will render an image of the grid.
A built in function - "grid read_file(char* f)" - will read a comma-delimited file to create a grid.
A built in function - "grid random_grid(int n, int m, int a)" - will return a random *nxm* grid with an alphabet of *a* symbols.

Cell symbols can be any ASCII character in 0-9 and A-Z.

Data Types

CAL will support the following primitive data types:
int - an integer
char - a character
bool - a boolean
Grid - an nxm array of characters that represents a cellular automata grid
Rule - an object that provides instructions on how to manipulate each cell in a grid based off it's neighboring 8 cells;

Declarations and Assignment

CAL is a strongly typed language. Types must be explicitly stated for each variable in the usual way:
[variable_type] variable_name;
[variable_type] variable_name = variable_value;

Initializing Grids

Grids can be either be created by hand, read from a file, or created randomly. The read_file and random_grid functions are mentioned above. Here is an example to create a grid by hand:

grid g = [A, B, 2,B; B, A, 1, C; A, A, A ,A];

This creates the following grid, g:

A	B	2	B
B	A	1	C
A	A	A	A

The ';' means to start a new line on the grid. The ',' means to start a new cell on the same line.

Initializing Rules

A rule explains if a cell should be changed given the states of its 8 neighboring cells.
rule <rule_name> = "
    [starting_symbol] : [top_left, top_middle, top_right, left, right, bottom_left, bottom_middle, bottom_right] => [ending_symbol]
    ";
Question marks - '?' - can be used as wild cards to include all symbols.
Here is an example of creating a rule with an alphabet of two symbols - {A,B}:
rule r = "
    A : [A , B , A; ? , ?; ?, ?, A] => B

A : [B , B , A; ? , ?; ?, ?, A] => A
B : [A, A, A; ?, ?; ? , A , ?] => A “;

Now suppose this is a 3x3 segment of a much larger grid:

A	B	A
B	A	B
B	A	A

When this rule is applied to the middle cell (the blue shaded A in the middle) , it will match with the first rule - *A : [A , B , A; ? , ?; ?, ?, A] => B.* Thus, the middle cell will be changed from an A to a B.

Operators
CAL will support the usual operators for integers:
int a;
int b;
a+b //addition;
a-b //subtraction;
a*b //multiplication;
a/b //integer division;
a>b //greater than;
a<b //less than;
a==b //equals;
a>=b //greater than or equal to;
a<=b //less than or equal to;
a%b //modulo;

Statements and Control Flow
CAL will have similar statements and control flow as the C language:
-a statement is terminated by “;”
- *if* /*else* control blocks as well as *for* and *while* loops will be included in CAL

Interesting Language Example
The following is a random number generator created with the the Rule 30 cellular automata by Stephen Wolfram: http://en.wikipedia.org/wiki/Rule_30

grid g = [0, 0, 0, 0, 0, 0; 1, 0, 1, 1, 0, 0; 0, 1, 0, 1, 1, 0; 1, 0, 0, 0, 0, 1; 0, 1, 1, 0, 0, 0];

rule rule30 = “
    ? : [0, 0, 0, ?, ?, ?, ?, ?] => 0

```
        ? : [0, 0, 1, ?, ?, ?, ?, ?] => 1
        ? : [0, 1, 0, ?, ?, ?, ?, ?] => 1
        ? : [1, 0, 0, ?, ?, ?, ?, ?] => 1
        ? : [1, 0, 1, ?, ?, ?, ?, ?] => 0
        ? : [1, 1, 0, ?, ?, ?, ?, ?] => 0
        ? : [1, 1, 1, ?, ?, ?, ?, ?] => 0 “
for (int i = 0; i < 50; i++){
        run(g, rule30);
}

display(g);
```