Developing a Unidimensional Cellular Automata

Original Author(s): Koen Koning & Michael Lees
Edited by: Johannes Blaser, Frederick Kreuk & Florine de Geus

Assignment Objectives
In this assignment you will develop a unidimensional cellular automata. Before starting on this assignment you should have a decent understanding of how a unidimensional CA works. See the slides or your favourite search engine for more details. The assignment itself has four objectives:

1. Understand how Python and the used libraries work with regards to CA
2. Develop a model of a unidimensional CA that supports different values for k and r.
3. Combine your model with the provided GUI to visualise and interpret qualitative behaviours of unidimensional CA.
4. Use your model to explore and build an understanding of unidimensional CAs.

Note: Please refer to the CA1 assignment for the submissions rules and program requirements. Failure to comply with them can mean exclusion from the grading process.

Detailed Description

At the end of this assignment you should have a functioning unidimensional cellular automata that can use a range of k (i.e. the alphabet size) and r (i.e. the neighbourhood) values. Your assignment should also automatically produce a graph showing average cycle lengths for elememtary cellular automata.

With $k = 3$ and $r = 1$ we have an input alphabet (or rule table size) of 3^3. The total number of possibile rule tables is then $3^{3^3} = 7,625,597,484,987$. A *rule number* specifies a specific instance of a particular rule table from **all possible rule tables**. The rule number corresponds to the output configuration of the rule table (the input sequence being fixed in decreasing order). When running the model the maximum rule number can be calculated (from the values of k and r) and should be displayed. You can then specify a rule number in decimal format. To interpret rule 34 we need to convert it to a number in base-k of length $3^3 = 27$ (note that $k = 3$ and $r = 1$ still hold). We then have the following conversions (the subscript specifies the base):

$$34_{10} = 1021_3$$

This yields the following rule table: `[0 ... 0 1 0 2 1]`. Note that there are 23 leading zeroes.

Base conversion can be tricky. Make sure you understand how it works thoroughly! You will use it in the following assignments as well so it is important to get this part right. If you are still struggling have a look at this page.

You are free to design the program any way you would like granted that it performs (well) on Linux platforms. We do however provide you some skeleton code to get started. We strongly recommend using the structure specified therein. The following section details how to proceed with the provided skeleton code.

The design

You should get started on the function `build_rule_set`. This function should – given a decimal rule number n – return a list of length $k^{2 \cdot r+1}$ representing the rule table for the CA. Using the example of rule number 34 we would then have $n = 34$, $k = 3$, and $r = 1$. The function should then return the following list: `[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 2 1]`. Note that the length of the array is equal to $3^{2 \cdot 1+1} = 3^3 = 27$.

You now have the rule set that determines the behaviour of the CA. Each step then uses this rule table to determine what the next value of a cell will be given a neighbourhood. To recap: the first zero in the rule implies that given a neighbourhood of `[2 2 2]` the next value of the middle cell would be nought. Remember that the list is stored in **decreasing order**.

The `step` method of the model will determine the new neighbourhood for **every** cell. For this the `check_rule` function is used. The latter ought to determine the new value for a neighbourhood. This value should be returned.

In Summary

Building on the skeleton code you mainly need to implement 4 functions to make the simulations run:

- decimal_to_base_k
- build_rule_set
- check_rule
- setup_initial_row

Before getting started it is strongly advised to have a clear understanding of what you are doing and how the program works. Comments in the provided skeleton should shed some light on the ideas behind some of the algorithms so be sure to read them.

Verification and Analysis

After you have implemented your model you ought to ensure it to be correct. There is a simple way to do so: compare your output to output (from other implementations) that you know to be correct. If you set $k = 2$ and $r = 1$ you have the classic Wolfram rules. There are many visualisations of this out there on the web – most notably by Wolfram Alpha. Compare against these for a number of elementary rules. Pick rules that show some interesting behaviour in different classes. Some good rules to start with could be:

- 30
- 110
- 184
- 220

Measurement and Testing

The purpose of the assignment is to perform experiments on your model and your model is merely a means to an end. In this section we describe what experiments and analysis you ought to perform on your model. Now you have verified that your code creates correct results you can again look at running some experiments.

An interesting experiment is to analyse the average cycle lengths of all 256 Wolfram rules (i.e. with $k = 2$ and $r = 1$). Of course you will have to limit how long to run your simulations for. More specifically you will have to set a limit after which you **assume** there to be no cycles. Of course a unidimensional CA is in principle finite but some rules have execution times that are outside of the range of feasible computability. Hence a value like 10^4 or 10^6 is a good cut-off point after which to assume no cycles occur and still maintain acceptable performance. In development smaller values may be desirable due to their significantly reduced execution time.

Note: as with any experiment it is absolutely **imperative** you mention your experimental set-up. You should clearly report on the tested system sizes, repetitions, or any other experimental parameter you've used. **Also include motivation behind your decisions.**

Also important are the initial conditions of your model. **Make sure** your model supports random and single seeds via the toggleable parameters (i.e. use Boolean types (`bool` in Python)). Some initial conditions can quickly lead to a cycle (or quiescent states) or have very different *transient lengths* so make sure to repeat experiments enough such that your results are reliable and statistically significant.

For your plots plot the rule numbers $[0, 255]$ alongs the X-axis and the (average) cycle lengths along the Y-axis. For your experiments you can think of the following:

- Sorting rule numbers along the X-axis in the order of their Wolfram class. Can you distinguish clear differences and is it as you would expect?
- If you can think of additional experiments you can perform using your model you should do so. This would – of course – be beneficial for your grade.

Deliverables and Grading

You ought to submit your program code and resulting figures to Canvas. The submissions requirements are specified in the CA1 assignment description. If you have a figure you should include **any and all** scripts that were used to generate the figures. Running these should produce the exact same figures that are in your submissions.

Your figures should show your analysis including labeled axes and a caption of **200 to 300 words**, discussing your results. Note that it is unnecessary (and not permitted) to submit a full written report, a clearly readable PNG image will suffice. You may include the caption as a separate text document, or include it in the image itself.

Your grade is primarily based on the correctness (and neatness) of your code as well as the validity and clarity of the resulting figure and description. Pay attention to the things mentioned above when working on this assignment.

As with any assignment your code will be checked for plagiarism using special-purpose software.