Computation versus Simulation

Is 'compute' a transitive or intransitive verb? That is, must one compute something or can one just compute? This distinction (or rather the failure to make it) underlies much of the confusion concerning 'computing' with cellular automata. Unfortunately, the OED doesn't help us on this question; both usages date from the 17th century. But Turing computable clearly describes the computation of something, while much of the CA literature concerns computing (verv intrans). This note is written from the Turing perspective.

A one dimensional cellular automata of radius r with S states is determined by a function f from S^{2r+1} to S. It acts on an infinite tape with S symbol by simultaneously updating each cell by applying f to the segment of length 2r+1 centered on that cell. f determines a global transition function G_f from S^Z to S^Z given by mapping $\sigma \in S^Z$ to $G_f(\sigma)$, the sequence resulting from this simultaneous updating. (Wolfram has assigned names to the 256 2-state 1-dimensional ca of radius 1.)

(There are natural generalizations to higher dimensions but they introduce no new phenomena for the discussion below.)

The notion 'supporting universal computation' is ambiguous in the literature. This ambiguity arises in my view from insufficient attention in the cellular automata literature to the two aspects of computing a function on the natural numbers: 1) fixing a coding and decoding of the natural numbers into the space in which the computation takes place (input/output rules) and 2) describing the steps in the computation.

Much of the cellular automata literature focuses on 2) - the simulation of one CA by another. But the notion of computation in the standard Turing sense depends essentially on the first.

So in the spirit of fom /FOM let me define two terms.

1. A universal computing automata (UCA) is a ca (WITH input/output rules) that can compute every partial recursive function).

2. A universal cellular automata (uca) is one that can simulate every other ca (and therefore every Turing machine).

I will argue that it is not evident that every universal cellular automata is a universal computing automata, because 'simulation' tends to ignore i/o conventions.

To be more precise, even if there are two-state universal cellular automata (and if Wolfram/Cook are correct there is one of radius 1), this does not automatically imply there is a two-state universal computing automata. (I'll elaborate on this point below once some notation is established.)

UNIVERSAL COMPUTATION: The most naive notion would be to code n by placing n contiguous 1's and the rest 0's on an infinite tape. If the configuration evolved under the ca C from this initial configuration to m contiguous 1's and the rest 0's we would say that C computed the value m with input n; if not, n is not in the domain of the partial function calculated by C. (This has some difficulty if we actually want to view it as a computer; we start the machine on a sequence of n's. After awhile, all we can see are m ones in a sea of 0's. How do we know this state persists?)

To emphasize, a universal Turing machine is one which computes every partial recursive function. As above a universal computing automata is one which computes every partial recursive function. This requires a definition of how to read output since CA's don't halt. That is, some description should be given of how to read an output from the eventual behavior of the CA. Thus, it is not enough to be able to simulate the actions of a given universal computing automata but an analysis of input/output is essential.

The most general approach can be summarized as follows. There should be two 'effective' functions f and g which map numbers to an infinite tape and back. Now a function ϕ of numbers is computed by a cellular automata if $\phi(x) = g(T(f(x))$ where T is the ordinary action of a cellular automata.

There are several complications with such generality. What does T(y) – the result of applying a ca mean? After all, ca' s don't halt. Even if this is overcome, what does it mean to say that g is an effective function if its domain is all infinite sequences.

The most common way to deal with this is to posit the existence of a quiescent (absorbing, blank) state. An infinite sequence is a 'configuration' if all but finitely many cells are blank. The papers of [4] and [1] share the convention that the input, steps in the computation, and output all take place on configurations. That is, f and g above map into and from the countable collection of configurations. Then, f and g are 'effective' means they are recursive in the normal sense of functions between finite sequences. We refer to this specification of f, g as an I/O convention. Other offers consider more general f and g.

SIMULATION The notion of simulation of cellular automata has been made precise in several places. Here is one adapted from (e.g. [3], [4]).

Let Φ_d be the collection of global transition functions for d dimensional ca (with arbitrary set of states). For $F \in \Phi_d$, F^k denotes the k-fold iteration of F.

Let Z_1 and Z_2 be d dimensional cellular automata with sets of configurations C_1 and C_2 (again, C_1 and C_2 are the sequences which have only finitely many non-quiescent states) and global transition functions F_1 and F_2. Let k_1, k_2 be positive integers. Then Z_2 simulates Z_1 in k_2/k_1 times real time if there exists an effectively computable injective mapping G from C_1 to C_2 such that

$$F_2^{k_2}(G(c)) = G(F_1^{k_1}(c)).$$

Smith [3] proves that one can trade off radius for number of states -in both directions. One can conclude that if there is any cellular automata which can compute all partial recursive functions there is one with only 2 states (but possibly very large radius.) The Wolfram-Cook claim for rule 110 is that there is one with 2 states and radius 1.

RELATION TO COMPUTABILITY: These simulation arguments disregard input–output conventions. The effect of the function G mapping configurations to configurations must be taken explicitly into account to make precise claims about computability. The basic difficulty with a two state machine (NO BLANKS) is, 'Where does the input/output end and the enveloping sea of zeros's begin?'

THE DENSITY PROBLEM: To emphasize the importance of fixing an I/O convention let me describe a family of results concerning one important problem This includes one result which at first blush contradicts the Wolfram-Cook claim.

A long-standing cellular automata problem is to recognize the density of a sequence of 0's and 1's. That is, 'classify all configurations of size n for density r', which means the ca evolves to all 1's on a finite sequence if the ratio of 1's to 0's is at least r and to all 0's if the ratio of 1's to 0's is less than r.

There is a large literature trying to find ca's which can accomplish this task. 'Genetic algorithms' attempt to modify a given (family) of ca's based on experimental results to 'breed' one which can solve the problem. Much of the work is with r equal 1/2.

Land-Belew [5] prove

Theorem: For a given neighborhood radius r, density rho, and sufficiently large lattice size, (depending on rho, and r) there is no 2 state CA of radius r which correctly classifies all configurations of size n for density rho.

Since this is clearly a recursive problem, at first glance, this appears to refute the assertion that there can be any 2-state CA which is computationally universal - contra Wolfram's claims vis-a-vis rule 110. But, there is an implicit reliance on an I/O convention. The input is a 'finite' but arbitrarily long string of 0's and 1's. Analysis of the proof suggests that the authors assume that the rest of the tape contains only 0's. If the machine evolves to n consecutive 1's the density is greater than r; if to all zero's, less than r. (Implicitly, the n-cells which we are focusing on can move on the infinite tape.)

On the other hand if we consider the problem to determine whether an input rational number q is greater than a fixed rational r, we would solve it with a 2-step automata by coding the problem in a k state machine (where k is large enough so we can easily construct a universal cellular automata) and then apply a procedure like Smith's to shift the problem to a 2 state machine (with some strange input convention) not at all resembling the natural one.

The density problem has an interesting follow-up which emphasizes again the need to fix i/o conventions. Rules 184 and 226 conserve the number of 0's and 1's and group the cells with the same value. So using a different OUTPUT rule [6] show these two rules do classify for density 1/2. Later work by e.g. Hendryk Fuks [2] solves the problem (for 1/2 and others for rational density) by composition of ca's to get back the original output convention.

WOLFRAM'S CLASSIFICATION: Based on computer simulations Wolfram presented in several papers conjectured classifications of cellular automata into 4 types. In [8] Wolfram distinguishes the 4 classes of cellular automata by the evolution of the pattern generated by applying a cellular automaton to a finite input. We quote from page 161.

1. Pattern disappears with time.
2. Pattern evolves to a fixed finite size.
3. Pattern grows indefinitely at a fixed rate.
4. Pattern grows and contracts with time.

Wolfram suggests that these dynamic considerations should be related to the computational complexity of the ca. Indeed this contention is one of the most interesting aspects of his book.

There are several difficulties. First, this classification is not precise. As ([1]) show, it can be made to make sense as a classification of the action a ca on a particular initial configuration. But even probabilistically, it cannot be extended to a classification of the action of ca's on finite input. Ishii [7] shows one can make sense of it as classification on arbitrary inputs. Culik et al [4]construct a hierarchy by looking at the worst case behavior.

Second, while this classification makes sense only when considering arbitrary (even arbitrary infinite input) the ordinary study of computation concerns only 'good' finite inputs. (In fact, much of interest in [4] and [1] deals with the behavior of ca's on arbitrary finite input.

The study of this problem led to another example of the importance of I/O conventions. Baldwin and Shelah showed how describe a universal automata whose pattern grew monotonically in size (add a marker that says 'this cell has been hit'); this contradicted an assertion in [4]. The distinction was that the output function g describe above is required by Culik et al to be injective while it was many one in Baldwin-Shelah.

References

[1] J.T. Baldwin and S. Shelah. On the classification of cellular automata. *Theoretical Computer Science*, 230:117–129, 2000.

[2] Henryk Fuks. Solution of the density classification problem with two cellular automata rules. *LANL comp-gas/9703001*, 1997.

[3] Alvin Ray Smith III. Cellular automata complexity trade-offs. *Information and Control*, 18:466–482, 1971.

[4] S. Yu K. Culik III, L. Hurd. Computation theoretic aspects of ca. *Physica D*, 45:357–378, 1990.

[5] Mark Land and Richard K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74:5148–5150, 1995.

[6] M. Sipper M. S. Capcarrere and M. Tomassini. Two-state, r=1 cellular automaton that classifies density. *Phys. Rev.Lett.*, 77:4969, 1996.

[7] S.Ishii. Measure theoretic approach to the classification of cellular automata. *Discrete Applied Mathematics*, 39:125–136, 1992.

[8] Stephen Wolfram. Computation Theory of Cellular Automata. In *Cellular Automata and Complexity*, pages 159–202. Addison Wesley, 1994. Originally published: Communications in Mathematical Physica 96 (1984) 15-57.

John Baldwin, Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago jbaldwin@uic.edu