Chapter 10
Computational Technosphere and Cellular Engineering

Mark Burgin

Department of Mathematics, University of California, USA

Abstract. The basic engineering problem is to build useful systems from given materials and with given tools. Here we explore this problem in the computational technosphere of computers, smartphones, networks and other information processing and communication devices created by people. The emphasis is on construction of different kinds of information processing automata by means of cellular automata. We call this engineering problem cellular engineering. Various types and levels of computing systems and models are considered in the context of cellular engineering.

Keywords: cellular automaton, computational equivalence, engineering, modeling, construction, model of computation, grid automaton.

1 Introduction

Stephen Wolfram [11] suggested the *Principle of Computational Equivalence*, which asserts that systems found in the natural world can perform computations up to a maximal ("universal") level of computational power, and that most systems do in fact attain this maximal level of computational power. Consequently, most systems performing recursive computations are computationally equivalent in general and equivalent to cellular automata in particular. Here we consider a technological counterpart of this Principle, which is related not to nature but to the *technosphere* created by people. The technosphere is the world of all technical devices. In it, computers and other information processing systems play the leading role. Taking all these devices, we obtain the *computational technosphere*, which is an important part of the technosphere as a whole. The computational technosphere has its own Principle of Computational Equivalence. It is called the *Church-Turing Thesis.* There are different versions of this Thesis. In its original form, it states that *the informal notion of algorithm is equivalent to the concept of a Turing machine* (the Turing's version) or that *any computable function is a partial recursive function* (the Church's version). The domineering opinion is that the Thesis is true as it has been supported by numerous arguments and examples. As a result, the Church-Turing Thesis has become the central pillar of computer science and implicitly one of the cornerstones of mathematics as it separates provable propositions from those that are not provable. In spite of all supportive evidence and its usefulness for proving various theoretical results in computer science and mathematics, different researchers, at first, expressed negative opinion with respect to validity of the Church-Turing Thesis, and then build more powerful models of algorithms and computations, which disproved this Thesis. It is possible to find the history of these explorations in [6]. Here we go beyond the computational technosphere, suggesting the *Technological Principle of Computational Equivalence* for the whole technosphere. It asserts:

For any technical system, there is an equivalent cellular automaton.

This principle also has a constructive form:

For any technical system, it is possible to build (find) an equivalent cellular automaton.

Here we consider only the computational form of the *Technological Principle of Computational Equivalence*. It is expressed as the *Computational Principle of Technological Equivalence*:

For any information processing system, it is possible to build (find) an equivalent cellular automaton.

Note that in this Principle cellular automata are not restricted to classical cellular automata. There are much more powerful cellular automata. For instance, inductive cellular automata can solve much more problems than classical cellular automata or Turing machines. Building technical systems is an engineering problem. That is why in Section 2, we discuss computational engineering, which is rooted in the work of von Neumann who used a special kind of computational engineering, or more exactly, cellular engineering, for building self reproducing automata [28]. He also demonstrated that construction of complex systems using cellular automata allows one to essentially increase reliability of these systems. However, to be able to rigorously demonstrate validity of the *Computational Principle of Technological Equivalence*, as well as of the *Technological Principle of Computational Equivalence* and Wolfram's *Principle of Computational Equivalence*, it is necessary to ascribe exact meaning to terms used in these principles. That is why in Section 3, we introduce and analyze different types of computational and system equivalence. In Section 4, we demonstrate possibilities of cellular engineering in modeling and construction, giving supporting evidence for the *Computational Principle of Technological Equivalence.* Some of these results were obtained in [7], while other results are new.

2 Computational Engineering

It is possible to describe an engineering problem in the following way. Given working materials and tools for operation, build/construct a system (object) that satisfies given conditions. Here we consider a specification of such a problem for computational (information processing) systems. Thus, we have the following initial conditions:

- A class **K** of computational (information processing) automata is given.
- A class **H** of computational (information processing) automata is provided.
- A set **A** of composition operations is made available.
- A type ϕ of automata equivalence is offered.

Task 1: For any automaton H from **H**, construct an automaton K from **K** by means of operations from **A**, such that K is ϕ-equivalent to H.

Task 2: For any automaton H from **H**, construct an automaton A ϕ-equivalent to H using operations from **A** and automata from **K** as the building material for operations from **A**.

Note that in the second case, the automaton A does not necessarily belong to the class **K**.

The area where such problems are solved is *computational engineering*. When the class **K** consists of cellular automata, i.e., we construct using cellular automata as the construction media, the construction problem is in the scope of *cellular engineering* introduced and studied in [7]. Another basic problem of cellular engineering is construction of different automata, such as pushdown automata, Turing machines and others, using cellular automata as building bricks, blocks and modules. In this case, the result of construction is a grid automaton [6] in a general case and only in some cases it can be a cellular automaton, which is a particular case of grid automata.

Note that there is one more type of computational engineering problems. In it, we have the following initial conditions:

- A class **K** of computational (information processing) automata is given.
- A set **A** of composition operations is made available.
- A goal σ is offered.

Task 3: Using operations from **A** and automata from **K** as the building material for operations from **A**, construct an automaton A that allows one to achieve the goal σ. Usually such a goal σ represents realization of certain functions and satisfaction of selected conditions.

There are three main types of cellular engineering:

- *Process cellular engineering* is aimed at building a cellular automaton to reproduce, organize, model or simulate some process.
- *Function cellular engineering* is aimed at building a cellular automaton to reproduce, organize, model or simulate some function.
- *System cellular engineering* is aimed at building a cellular automaton to reproduce or model some system with its subsystems, components and elements.

Traditional engineering problems for cellular automata are mostly related to process cellular organization or reproduction, that is, how to get a process with necessary characteristics in a cellular automaton. Only sometimes functions are modeled like when cellular automata are used to model functioning of a Turing machine. System cellular engineering reproduces (models) a system with some level of detailing. For instance, it is possible to represent a system at the level of its elements or at the level of its components.

The area of cellular automata can be divided into three big subareas: *CA science, CA computation*, and *CA engineering.* CA science studies properties of cellular automata and particular, their dynamics or how they function. CA computation uses cellular automata for computation, simulation, optimization, and generation of evolving processes. CA engineering is aimed at constructing different devices from cellular automata. All three areas are complementary to one another.

Ideas similar to the concept of cellular engineering were also discussed by Deutsch in the form of *constructor theory* and *verifiable metaphysics* [10].

Cellular automata are the simplest uniform models of distributed computations and concurrent processes. Grid automata are the most advanced and powerful models of distributed computations and concurrent processes, which synthesize different approaches to modeling and simulation of such processes [4, 6].

Informally a *grid automaton* is a system of automata, which are situated in a grid and called nodes. Some of these automata are connected and interact with one another. It is possible to find formal definitions and elements of the theory of grid automata in [4, 6].

Cellular automata are special cases of grid automata although, in general, grid automata are non-uniform. Our goal is not to substitute cellular automata by grid automata, but to use cellular automata as the basic level for building hierarchies of grid automata. The reason for doing this is to reduce complexity of the description of the system and its processes. For instance, computer hardware has several levels of hierarchy: from the lowest logic gate level to the highest level of functional units, such as system memory, CPU, keyboard, monitor, printer, etc. In addition, as Clark writes (cf. [15]), all good computer scientists worship the god of modularity, since modularity brings many benefits, including the all-powerful benefit of not having to understand all parts of a problem at the same time in order to solve it. That is why one more goal of this paper is to introduce modularity into the realm of cellular automata, making possible to get better understanding and more flexible construction tools without going into detailed exposition of the lower levels of systems. As a result, we develop a computing hierarchy based on cellular automata.

Cellular engineering is an approach complimentary to evolutionary simulation and optimization. Evolutionary simulation is aimed at modeling complex behavior by simple systems, such as cellular automata. Evolutionary optimization is aimed at improving systems by simple means of automata, such as cellular automata, which imitate natural evolutionary processes. Cellular engineering is aimed at constructing complex systems using simple systems, such as cellular automata. In evolutionary processes, systems are evolving subject to definite rules. In engineering, systems are purposefully constructed according to a designed plan.

3 Types of System Equivalence, Modeling and Construction

When researchers discuss equivalence of different models of computation, they are, as a rule, dealing only with one type of equivalence - functional equivalence. The reason is that initially computation performed only computation of functions. Later with an advent of electronic computers, computation enormously expanded its domain but the initial imprinting continues to influence computer science.

At the same time, there is a variety of different types and kinds of equivalence between computational models, automata, software systems, information processing systems and computer hardware. We consider only some of them:

1. Functional equivalence.
2. Linguistic equivalence.
3. Computational equivalence.
4. Structural equivalence.
5. Complexity functional equivalence.
6. Local functional equivalence.
7. Operational or process equivalence.
8. Local operational equivalence.

Let us consider definitions of these types. Two classes of algorithms/automata are *functionally equivalent* if they compute the same class of functions. Two classes of algorithms/automata are *linguistically equivalent* if they compute the same class of languages. Two classes of algorithms/automata are *computationally equivalent* if what is possible to compute in one class it is also possible to compute in the other class. Two classes of algorithms/automata are *operationally or processually equivalent* if they generate the same class of computational processes. Two classes of algorithms/automata are *locally operationally equivalent* if they can perform the same class of computational operations. Two classes of algorithms/automata are *functionally equivalent with respect to complexity* if they compute the same class of functions with the same complexity. Two classes of algorithms/automata are *functionally equivalent with respect to completion* if they compute the same class of functions with the same (level of) complexity. All these definitions describe direct types of equivalence. At the same time, there are more advanced but also useful transcribed types of equivalence. *Transcribed equivalence* includes coding and decoding. For instance, an automaton or a software system B is *functionally equivalent with transcription* to an automaton or a software system A if there are two automata (software systems) C, D, F and G such that for any input X to A, we have $A(X) = D(B(C(X)))$ and for any input Y to B, we have $B(Y) = G(B(F(X)))$. In these processes, C and F are coders of information, while D and G are decoders of information. Functional and processual types of equivalence bring us to the concept of modeling.

Definition 1. It is possible to model an abstract automaton A by a cellular automaton C if there is a configuration W of cells from A and a system R

of states of cells from W such that after initializing these states, the cellular automaton C works as the automaton A.

This modeling relation is related either to process cellular engineering or to function cellular engineering. It is necessary to remark that modeling relation plays an important role not only in information processing systems or other technical systems but also in all life processes and living systems [16, 12]. In some cases, individual cellular engineering allow us to perform cellular engineering for classes of automata.

Definition 2. It is possible to *model* a model $\mathbf{M}$ of computation in a class $\mathbf{C}$ of cellular automata if it is possible to model any automaton A from $\mathbf{M}$ by some cellular automaton C from $\mathbf{C}$.

There are different types of modeling.

Definition 3. An abstract automaton A is called programmable in a cellular automaton C if there is a configuration W of cells from A and a system R of states of cells from W such that after initializing these states, the cellular automaton C works as the automaton A, that is, with the same input, C gives the same result as A.
This is a *function cellular engineering.* It is defined by the functional equivalence.

We remind [9] that there are two kinds of functional modeling: direct and transcribed. *Direct functional modeling* of an automaton or a software system A by an automaton or a software system B means that given any input X to A, it either does give any result or gives the same result as the automaton (software system) B with the same input.

Transcribed functional modeling includes coding and decoding [9]. Namely, an automaton or a software system B allows transcribed functional modeling of an automaton or a software system A if there are two automata (software systems) C and D such that for any input X to A, we have $A(X) = D(B(C(X)))$. In this process, C is the coder of information, while D is the decoder of information. The process of transcribed functional modeling is described by the diagram in Fig. 1.

As in a general case, we can realize function cellular engineering for classes of automata.

$$\begin{tikzcd}
V \arrow[r, "B"] & W \\
X \arrow[u, "C"] \arrow[r, "A"] & Y \arrow[u, "D"']
\end{tikzcd}$$

Fig. 1. The process of transcribed functional modeling

Definition 4. A model of computation **M** is called *programmable* in a class **C** of cellular automata if any automaton A from **M** is programmable in some cellular automaton C from **C**.

Definition 5. An abstract automaton A is called *constructible* in a cellular automaton C if there is a configuration W of cells from A and a system R of states of cells from W such that after initializing these states, the cellular automaton C works as the automaton A. and to each structural component D of A some part B of the automaton C is corresponded in such a way that B works as D.

This gives us the construction relation related to *system cellular engineering*. It is defined by the structural equivalence.

Note that both modeling relation "A models B" and construction relation "A is constructed in B" are special cases of the fundamental triad [5].

Definition 6. A model of computation **M** is called *constructible* in a class **C** of cellular automata if any automaton A from **M** is constructible in some cellular automaton C from **C**.

To construct definite devices, we need elements from which we construct and algorithms how to do this. There are three main element types (in information typology), which correspond to the three main types of information operations described in [3]:

- Computational elements or *transformers*.
- Transaction elements or *transmitters*.
- Storage elements or *memory cells*.

There are three element types (in dynamic typology), which correspond to their dynamic:

- Elements with a fixed structure.
- Reconfigurable elements.
- Switching elements.

Elements with a fixed structure have the same structure during the whole process. *Reconfigurable elements* can change their structure during the process. *Switching elements* tentatively change their structure in each operation. There are three element types of memory cells: *read-only cells, write-only cells*, and *two-way cells*, which allow both reading and writing.

4 Construction of Information Processing Systems with Cellular Automata

Let us consider a model of computation **M** that has a universal automaton U.

Theorem 1. A model of computation **M** is programmable in a class **C** of cellular automata if and only if a universal automaton U is programmable in some cellular automaton C from **M**.

Note that here the transcribed equivalence is used because usually universal automata, e.g., universal Turing machines, model other automata from the same class, e.g., other Turing machines, only with transcription [6].

Corollary 1. A model of computation **M** is programmable in a cellular automaton C if the automaton U is programmable in C.

For illustration, we give here a well-known result in the theory of cellular automata.

Theorem 2. The class **T** of all Turing machines is programmable in the class C_1 of one-dimensional cellular automata.

Lemma 1. If a class **A** of automata is programmable in a class **C** of automata and a class **C** of automata is programmable in a class **B** of automata, then the class **A** is programmable in the class **B**.

It is known (cf., for example, [6]) that any class of recursive algorithms, such as partial recursive functions, random access machines (RAM) or Minsky machines, as well as any class of subrecursive algorithms, such as recursive functions, pushdown automata or context free grammars, is programmable in the class **T** of all Turing machines. Thus, Lemma 1 and Theorem 2 give us the following result.

Theorem 3. Any class of recursive algorithms (any class of subrecursive algorithms) is programmable in the class C_1 of one-dimensional cellular automata.

Corollary 2. An arbitrary pushdown automaton is constructible in the class C_2 of two-dimensional cellular automata.

Building a two-dimensional cellular automaton CA from multilevel finite automata [7], it is possible to prove the following result.

Theorem 4. A two-dimensional cellular automaton can realize any finite grid of connections between nodes in a grid automaton G.

To realize all these types of elements in cellular automata, multilevel finite automata described in [7] are used.

Corollary 3. If all nodes in a finite grid automaton G have a finite number of ports and are programmable (constructible) in one-dimensional cellular automata, then the automaton G is programmable (respectively, constructible) in a two-dimensional cellular automaton. Note that not any finite configuration is a finite automaton. For instance, at each step, a Turing machine is a finite configuration but it's not a finite automaton. Another example is when a node in a grid automaton can be an automaton that works with real numbers.

It is also possible to construct Turing machines in cellular automata.

Theorem 5. An arbitrary Turing machine with a one-dimensional tape is constructible in the class C_1 of one-dimensional cellular automata.

To prove this theorem, finite automata with inner structure are used.

Note that it is not the standard result that one-dimensional cellular automata can emulate a one-dimensional Turing machine. The standard result tells that an arbitrary Turing machine is programmable in the class C_1 of one-dimensional cellular automata. Theorem 5 establishes that an arbitrary Turing machine is constructible in the class C_1 . Constructability implies programmability but the converse is not true. For instance, any Turing machine with a two-dimensional tape is programmable in the class of Turing machines with a one-dimensional tape, but it is not constructible in this class.

As the class **T** has universal Turing machines, Theorems 1 and 5 imply the following result.

Corollary 4. The class **T** of all Turing machines with a one-dimensional tape is constructible in the class C_1 of one-dimensional cellular automata.

Global Turing machines or Internet machines introduced in [17] form a natural class of grid automata. An Internet machine is a finite grid automaton in which all nodes are Turing machines. Theorems 4 and 5 imply the following result.

Corollary 5. An Internet machine *IM* is constructible in the class **CA** of cellular automata. This implies the following result.

Corollary 6. The class **IM** of all Internet machines is constructible in the class **CA** of cellular automata.

Corollary 7. The class **T** of all Turing machines is constructible in the class **CA** of cellular automata.

In a similar way, it is possible to program inductive automata (inductive models of computation), which provide better modeling of contemporary computers and computer networks than traditional models, such as Turing machines [6].

Theorem 6. Any inductive Turing machine of the first order is programmable in the class **ICA** of inductive cellular automata.

Corollary 8. The class $\mathbf{IT}_1$ of all inductive Turing machines of the first order is programmable in the class **ICA** of inductive cellular automata.

Similar to Internet machines, it is useful to introduce inductive Internet machines, which also form a natural class of grid automata. Any Internet machine is a finite grid automaton in which all nodes are inductive Turing machines.

Theorems 4 and 6 imply the following result.

Corollary 9. Any inductive Internet machine *IM* is constructible in the class **ICA** of inductive cellular automata.

Computers and devices in global networks start processing data not only in the form of words, as conventional abstract automata do, but also more sophisticated structures. For instance, researchers forecast that future global networks will use graphs or heaps of soft protocol elements instead of multilayered protocol stacks used now [8, 2]. That is why it is important to represent not only words but other advanced structures using cellular automata. In addition, structures of computer hardware and software are much more sophisticated than linear structures of words.

Here is one result that demonstrates corresponding possibilities of cellular automata in modeling data structures.

Theorem 7. A two-dimensional cellular automaton can realize any finite graph or network.

In a similar way, cellular automata can realize many other data structures.

5 Conclusion

We discussed a new discipline – cellular engineering. Obtained results show how it is possible to construct and model sophisticated complex system using such relatively simple systems as cellular automata. The functional cellular engineering is one of the weakest forms, while the system cellular engineering is one of the strongest forms of cellular engineering.

Indeed, building a system with necessary properties solves the problem of creating a process with necessary features, while the latter solves the problem of constructing a function with necessary characteristics. Usually only functional cellular engineering has been considered, e.g., when cellular automata computed the same function as a Turing machine.

Modeling relation plays an important role in all life processes and living systems [16, 12]. Thus, it would be interesting to use cellular automata for modeling real living systems and not only some processes that resemble functioning of living systems as it is done in Artificial Life [1]. Moreover, in the context of pancomputationalism (cf., for example, [19, 11, 11, 13]) when the universe is treated as a huge computational structure or a network of computational processes which following fundamental physical laws compute (dynamically develop) its own next state from the current one, the *Technological Principle of Computational Equivalence* can be the base for constructor theory discussed by Deutsch in [10].

It is interesting to know that the method developed in [6] for construction of Turing machines and grid automata in cellular automata gives a formal representation of the old Internet idea [8] that any component with more than one network interface can be a router.

References

[1] Bornhofen, S., Lattaud, C.: Outlines of Artificial Life: A Brief History of Evolutionary Individual Based Models. In: Talbi, E.-G., Liardet, P., Collet, P., Lutton, E., Schoenauer, M. (eds.) EA 2005. LNCS, vol. 3871, pp. 226–237. Springer, Heidelberg (2006)

[2] Braden, R., Faber, T., Handley, M.: From protocol stack to protocol heap: Role-based architecture. ACM SIGCOMM Computer Communication Review 33(1), 17–22 (2003)

[3] Burgin, M.: Information Algebras. Control Systems and Machines (6), 5–16 (1997) (in Russian)

[4] Burgin, M.: Cluster Computers and Grid Automata. In: Proceedings of the ISCA 17th International Conference on Computers and their Applications, Honolulu, Hawaii. International Society for Computers and their Applications, pp. 106–109 (2003)

[5] Burgin, M.: Unified Foundations of Mathematics, Preprint Mathematics LO/0403186, 39 p. (2004), electronic edition: `http://arXiv.org`

[6] Burgin, M.: Superrecursive Algorithms. Springer, New York (2005)

[7] Burgin, M.: Cellular Engineering. Complex Systems 18(1), 103–129 (2008)

[8] Crowcroft, J.: Toward a network architecture that does everything. Comm. ACM 51(1), 74–77 (2008)

[9] Burgin, M.: Measuring Power of Algorithms, Computer Programs, and Information Automata. Nova Science Publishers, New York (2010)

[10] Deutsch, D.: Physics, Philosophy and Quantum Technology. In: Proceedings of the 6th International Conference on Quantum Communication, Measurement and Computing. Rinton Press, Princeton (2003)

[11] Fredkin, E.: Digital Mechanics. Physica D, 254–270 (1990)

[12] Kineman, J.J.: Modeling relations in nature and eco-informatics: A practical application of Rosenian complexity. Chemistry and Biodiversity 4(10), 2436–2457 (2007)

[13] Lloyd, S.: A theory of quantum gravity based on quantum computation. Preprint in Quantum Physics (2006) (arXiv:quant-ph/0501135)

[14] von Neumann, J.: Theory of Self-Reproducing Automata. 1949 University of Illinois Lectures on the Theory and Organization of Complicated Automata, Edited and completed by Arthur W. Burks. University of Illinois Press, Urbana (1966)

[15] Peterson, L.L., Davie, B.S.: Computer Networks: A System Approach. Morgan Kaufmann Publishers, San Francisco (2000)

[16] Rosen, R.: Life itself: A Comprehensive Inquiry into the Nature, Origin and Fabrication of Life. Columbia University Press, New York (1991)
[17] Van Leeuwen, J., Wiedermann, J.: Breaking the Turing Barrier: The case of the Internet. Techn. Report, Inst. of Computer Science, Academy of Sciences of the Czech. Rep., Prague (2000)
[18] Wolfram, S.: A New Kind of Science. Wolfram Media, Champaign (2002)
[19] Zuse, K.: Rechnender Raum. Friedrich Vieweg & Sohn, Braunschweig (1969)