The Blyth Institute
http://www.blythinstitute.org/

Wolfram's Complexity Classes, Relative Evolvability, Irreducible Complexity, and Domain-Specific Languages

Jonathan Bartlett
The Blyth Institute

Abstract

Stephen Wolfram observed four basic classes of complexity in his analyses of cellular automata based on their outcomes from a disordered initial state. Class 1 automata always arrived at a homogenous state no matter what their initial state was. Initial states in Class 2 automata have a finite sphere of influence for outcomes even if the computation was carried out an infinite number of steps. Class 3 automata are chaotic systems, where initial states do not have either a bounded sphere of influence nor do they produce predictable results. However, they do have stable statistical properties. Class 4 automata exhibit a hybrid of both the periodic behavior of Class 2 systems and the chaotic nature of Class 3 systems. More importantly, they are unpredictable both in their exact outcomes as well as well as in their statistical properties. These classes apply not only to cellular automata specifically, but to any type of programming language or system, for which the initial state is a program defined in a language.

Wolfram observed that the only types of systems known to be capable of universal computation were systems exhibiting Class 4 complexity. Therefore, while it may be possible to build a program on a Class 4 system which does not require its chaotic attributes, if a Class 4 automata is required for universal computation, then it is reasonable to suggest that a computation which requires a universal computer would also need to make use of the chaotic features which make it exhibit Class 4 complexity.

Because of the chaotic nature of Class 4 systems, there is not a smooth transition from changes in programming to changes in outcomes. Thus, systems which need to rely on Class 4 behaviors have great difficulty arising from natural selection because the chaotic mapping from the system's programming to the system's results prevents there from being a selectable path leading towards a solution. There is not yet a metric to measure evolvability in this way, though cyclomatic complexity may be a good starting point for research in this area.

In most programming systems, the most chaotic elements within those systems come from explicitly-controlled loops. Interestingly, while several evolutionary systems have been developed which utilize an underlying universal computer (such as Avida and GEMS), there have been no systems of which this author is aware where a loop control structure has been built from scratch using evolutionary algorithms, and the loop was required for the computation to be successful.

Such systems might still be evolvable, however. Many approach evolvability as an absolute measurement. However, the evolvability of a system depends both on the nature of the evolved system as well as the implementation language. A system may be complex in one language, requiring chaotic elements to implement it, while in another language it can be implemented using non-chaotic elements. While the complexity characterization of a system can determine its evolvability with regards to one implementation language, that does not prevent it from being more easily evolved via another language, such as a domain-specific language where features of the language are more closely mapped to the expected solution domain.

Therefore, a system which is unevolvable can be perfectly evolvable in another context. Therefore, designations such as "irreducible complexity" are relative designations, not absolute ones. Demonstrating the evolution of an "irreducibly complex" system would not invalidate either the concept or the designation, but rather point to a higher-level, domain-specific system which is guiding the evolution. Thus, "irreducible complexity" and similar ideas can be used as a detection method for higher-order evolutionary engines in operation within the genome. While further research is still required for an analytical method for detecting these systems, theory suggests that multiple, interacting feedback loops could only be evolved with the help of domain-specific systems.

References

- Bartlett, Jonathan L. 2006. "Metaprogramming and the Genome." *Occasional Papers of the BSG* 8:19-20.
- Crepeau, Ronald L. 1995. "Genetic Evolution of Machine Language Software." In: *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*. Tahoe City, CA.
- Lenski, Richard E. et al. 2003. "The evolutionary origin of complex features." *Nature* 423:139-144.
- Wolfram, Stephen. 1983. "Cellular Automata." *Los Alamos Science*, 9:2-21.
- Wolfram, Stephen. 2002. *A New Kind of Science*. Wolfram Media, Champaign, Ill.