On Semantics-to-Syntax Analyses of Algorithms

Yuri Gurevich
Microsoft Research
Redmond, WA, USA
gurevich@microsoft.com

Give me a fulcrum, and I will move the world.
Archimedes

Abstract: Alan Turing pioneered semantics-to-syntax analysis of algorithms. It is a kind of analysis where you start with a large semantically defined species of algorithms, and you finish up with a syntactic artifact, typically a computation model, that characterizes the species. The task of analyzing a large species of algorithms seems daunting if not impossible. As in quicksand, one needs a support, a rescue point, a *fulcrum*. In computation analysis, a fulcrum is a particular viewpoint on computation that clarifies and simplifies things to the point that analysis become possible.

We review from that point of view Turing's analysis of human-executable computation, Kolmogorov's analysis of sequential bit-level computation, Gandy's analysis of a species of machine computation, and our own analysis of sequential computation.

Keywords: algorithm, computation, semantics-to-syntax analysis, sequential algorithm, Turing, Kolmogorov, Gandy

1. Introduction

This paper is was conceived as an extended abstract, an appetizer of a kind, for a much longer article (Gurevich, to appear). But the paper is written after the fact and contains a number of observations not present in the longer article.

For brevity, classes of algorithms given by semantical constraints will be called *species of algorithms*. For example, the species of *sequential-time algorithms* consists of algorithms that execute step after step, and the species of *sequential algorithms* consists of sequential-time algorithms with steps of bounded complexity.

Semantics-to-syntax analysis of a species of algorithms seeks to find a syntactic definition of the species, typically a particular kind of machines that execute all and only the algorithms of the species. A classical example is analysis of human computation in (Turing, 1936-37). Often the original definition of the species is vague. Explicating it is a part of the analysis. In the process, the original species may be narrowed. For example, Turing stipulated that human computing is done by "writing certain symbols on paper" ruling out e.g. ruler-and-compass computations.

Discussion

Q[1]: In one place you speak about analysis of algorithms, in another about analysis of computation. Which is it?

A[2]: For our purposes, the analysis of algorithms and the analysis of computation are one and the same. We consider only algorithmic computations, and of course algorithms specify computations.

Q: In mathematical logic, theory of algorithms is the theory of computable functions.

A: Yes, but there may be much more to an algorithm than its input-output behavior. Algorithms perform tasks, and computing functions is a rather special class of tasks. By the way, for some

[1] Quisani, an inquisitive friend of ours.
[2] The author.

useful algorithms, non-termination is a blessing, rather than a curse; think for example of an algorithm that opens and closes the gates of a railroad crossing.

Acknowledgments

Many thanks to Andreas Blass for useful comments.

2. Alan Turing

Alan Turing analyzed human computing, even though the variety of human computations is mind boggling. What fulcrum supported the lever of Turing's analysis? We believe that his fulcrum was this:

Ignore what a human computer has in mind and concentrate on what the computer does.

In other words, Turing treated the idealized human computer as an operating system of sorts.

Discussion

Q: But Turing did not ignore the human computer's mind. He spoke about the state of mind of the human computer explicitly and repeatedly. Here is but one example. "The behaviour of the computer at any moment is determined by the symbols which he is observing, and his 'state of mind' at that moment."

A: Turing postulated that there are only finitely many states of mind which need be taken into account. They can be labeled, so that the computer needs to remember only the label of the current state of mind.

3. Andrei Nikolayevich Kolmogorov

Contrary to Turing, Andrei Nikolayevich Kolmogorov did not publish a detailed analysis of computation. His analysis is reflected in the proceedings of a 1953 talk to the Moscow Mathematical Society (Kolmogorov, 1953) and in a paper (Kolmogorov & Uspensky, 1958) with his student Vladimir Uspensky.

Kolmogorov might have intended to analyze all algorithms but the analysis in (Kolmogorov & Uspensky, 1958) is restricted to digital algorithms and does not apply to e.g. ruler-and-compass algorithms. Besides, the algorithms of his time still were sequential.

In the 1953 talk, Kolmogorov stipulated the following three principles.

K1. Sequentiality An algorithmic process splits into steps whose complexity is bounded in advance.

K2. Elementary steps Each step consists of a direct and unmediated transformation of the current state S to the next state S^*.

K3. Locality Each state S has an active part of bounded size. The bound does not depend on the state or the input size, only on the algorithm itself. The direct and unmediated transformation of S to S^* is based only on information about the active part of S and changes only the active part.

Kolmogorov and Uspensky analyzed sequential algorithms on the level of single bits. The syntactic definition of sequential algorithms is given by means of Kolmogorov-Uspensky machines. Kolmogorov's fulcrum seems to be this:

Computation is a physical process evolving in space and time.

Discussion

Q: The three principles say nothing about an algorithmic process being digital.

A: That is right. But the states of Kolmogorov-Uspensky machines are digital.

Q: Principle K2 does not look convincing. In one step, a sequential algorithm may perform state transformations — e.g. multiplications, divisions — that aren't so direct and immediate.

A: Kolmogorov and Uspensky do not allow such complicated transformations. Recall that they analyze sequential algorithms on the level of single bits where multiplication and division would have to be broken into a sequence of bitwise operations.

Q: Are the principles K1–K3 independent?

A: Not if you take into account that Kolmogorov and Uspensky analyzed computation on the level of single bits. (Of course our reply is bound to have a degree of informality: the principles K1–K3 aren't formal mathematical statements.) Since the active part of states is bounded, each such active part may be encoded with a binary string of a bounded length. Taking into account the elementary character of a single step of the algorithmic process in question, this implies a bound on the step complexity.

4. Robin Gandy

Gandy analyzed computation in his 1980 paper "Church's Thesis and Principles for Mechanisms" (Gandy, 1980). The computers of Gandy's time were machines, or "mechanical devices", not only humans, and that is Gandy's departure point.

> "Turing's analysis of computation by a human being does not apply directly to mechanical devices ... Our chief purpose is to analyze mechanical processes and so to provide arguments for ...
>
> **Thesis M.** *What can be calculated by a machine is computable*."

Mechanical devices can perform parallel actions, and so Thesis M "must take parallel working into account." But the species of all mechanical devices is too hard to analyze, and Gandy narrows it quite substantially.

> "**Thesis P.** *A discrete deterministic mechanical device satisfies principles I–IV below*."

Principle I asserts in particular that the states of any mechanical device can be adequately represented by hereditarily finite sets[3] and that there is a transition function F on hereditary finite sets such that, if x represents an initial state, then $Fx, F(Fx), \ldots$ represent the subsequent states. Gandy wants his analysis "to be sufficiently abstract to apply uniformly to mechanical, electrical or merely notional devices," so the term mechanical device is treated liberally.

Principles II are III are technical restrictions on states and the transition function. Principle IV generalizes Kolmogorov's locality constraint to parallel computations.

It seems to us that Gandy's fulcrum is the part of his Principle I described above:

> *The states of any mechanical device can be adequately represented by hereditarily finite sets, and there is a transition function F on hereditary finite sets such that, if x represents an initial state, then $Fx, F(Fx), \ldots$ represent the subsequent states.*

The fulcrum allowed Gandy to translate the bewildering world of mechanical devices into the familiar set-theoretic framework. Gandy's analysis led him to a synchronous parallel computation model.

[3] A set may contain other sets as elements. A set x is *hereditarily finite* if its transitive closure $\mathrm{TC}(x)$ is finite. Here $\mathrm{TC}(x)$ is the least set t such that (i) $x \in t$ and (ii) if $z \in y \in t$ then $z \in t$.

Discussion

Q: Machine parallelism does not have to be synchronous. It may be asynchronous. Isn't Thesis P too narrow.

A: It is, but asynchronous computations, as well as distributed, quantum, real-time computations, had not been developed at the time when Gandy wrote his paper. On the other hand, analog computing devices did play an important role at the time.

Q: Was Gandy the first to analyse synchronous parallel computation.

A: As far as we know, Gandy's analysis was the first semantics-to-syntax analysis of a large class of machine computations.

Q: Was Gandy's computation model the first parallel computation model?

A: No, certainly the circuit model had been used earlier. Complexity of circuits was studied already in (Shannon, 1949). Uniform families of circuits were introduced in (Borodin, 1977). But the logician Gandy most probably did not know the computer engineering literature.

5. Sequential algorithms

By the 1980s, there were plenty of computers and software. A problem arose how to specify software. Specifications can be thought of as higher-level algorithms. Accordingly, the analysis of algorithms on the level of single bits wasn't good enough anymore.

A semantics-to-syntax analysis of sequential algorithms was performed in (Gurevich, 2000). We mention extensions of that analysis below, in the next section, but this section is restricted to sequential algorithms. The analysis resulted in an axiomatic definition of sequential algorithms. A machine model for executing sequential algorithms on their natural levels of abstraction was given earlier; see e.g. (Gurevich, 1995). The machines became known as *abstract state machines* or ASMs.

Intuitively, as Kolmogorov pointed out, sequential algorithms compute in steps of bounded complexity. The formal definition of sequential algorithms consists of three axioms. It is instructive to compare them with Kolmogorov's principles K1–K3.

Axiom 1, the Sequential Time Axiom, is a part of principle K1. It asserts that an algorithm is a deterministic transition system given by a set of (some objects that we call) states, a subset of initial states and a state transition map.

Kolmogorov's principle K2 restricts the abstraction level of algorithms, and we go beyond it. We do not require that algorithms are digital; ruler-and-compass algorithms can be legitimate sequential algorithms. Also, the Gauss Elimination Procedure, operating on genuine real numbers, is a legitimate algorithm. Our axiom 2, the Abstract State Axiom, is closely related to the fulcrum that allowed us to pull off a semantics-to-syntax analysis of sequential algorithms. The fulcrum is this:

> *On the native level of abstraction of a sequential algorithm, its states can be faithfully represented by first-order structures of a fixed vocabulary in such a way that state transitions can be expressed naturally in the language of the fixed vocabulary.*

Axiom 2 itself goes a little beyond the fulcrum. Axiom 3, the Bounded Exploration Axiom, is the most technical of the three axioms. It generalizes Kolmogorov's principle K3 to algorithms on arbitrary abstraction levels.

The axiomatization allows us to prove the sequential ASM thesis of (Gurevich, 1995): every sequential algorithm can be step-by-step simulated by an appropriate sequential abstract state machine.

Discussion

Q: Your analysis cannot be sufficiently general because you restrict it to deterministic sequential algorithms. There are also nondeterministic sequential algorithms, e.g. nondeterministic finite automata.

A: The notion of nondeterministic sequential algorithm is a useful abstraction but it hides interaction (Gurevich, 2000, §9). An external intervention is needed to resolve nondeterminism. Recall Yogi Berra's famous nondeterministic sequential algorithm: "When you come to a fork in the road, take it!" The analysis in (Gurevich, 2000) is restricted to non-interactive sequential algorithms. More exactly, inter-step interaction with environment is allowed; it is intra-step interaction that is forbidden. Of course, intra-step interactive sequential algorithms are of much interest, and we analyzed them elsewhere (Blass & Gurevich, 2006-07; Blass, Gurevich, Rosenzweig, & Rossman, 2007).

6. Limitations on semantics-to-syntax analyses

In (Gurevich, 2012) we argued that the notion of algorithm cannot be rigorously defined in full generality, at least for the time being. The reason is that the notion is expanding. In addition to sequential algorithms, in use from antiquity, we have now parallel algorithms, interactive algorithms, distributed algorithms, real-time algorithms, analog algorithms, hybrid algorithms, quantum algorithms, etc. New kinds of algorithms may be introduced and most probably will be. Will the notion of algorithms ever crystallize to support rigorous definitions? We doubt that. Similarly, the species of machine algorithms evolves and may never crystallize.

However the problem of rigorous definition of algorithms is not hopeless. Not at all. Large and important strata of algorithms have crystallized and became amenable to rigorous definitions. In particular, the axiomatic definition of sequential algorithms was extended to synchronous parallel algorithms in (Blass & Gurevich, 2003) and to interactive sequential algorithms in (Blass & Gurevich, 2006-07; Blass et al., 2007). It was also used in to derive Church's thesis from the three axioms plus an additional Arithmetical State Axiom which asserts that only basic arithmetical operations are available initially (Dershowitz & Gurevich, 2008).

References

Blass, A., & Gurevich, Y. (2003). Abstract state machines capture parallel algorithms. *ACM Trans. on Computational Logic*, *4:4*, 578–651. (Correction and extension, same journal, 9:3 (2008), article 19)

Blass, A., & Gurevich, Y. (2006-07). Ordinary interactive small-step algorithms. *ACM Trans. on Computational Logic*. (Part I in Vol. 7:2 (2006), 363–419; Parts II and III in Vol. 8:3, articles 15 and 16 respectively)

Blass, A., Gurevich, Y., Rosenzweig, D., & Rossman, B. (2007). Interactive small-step algorithms. *Logical Methods in Computer Science*, *3:4*. (Parts I and II, papers 3 and 4 respectively)

Borodin, A. (1977). On relating time and space to size and depth. *SIAM Journal on Computing*, *6*, 733–744.

Dershowitz, N., & Gurevich, Y. (2008). A natural axiomatization of computability and proof of Church's thesis. *Bull. of Symbolic Logic*, *14:3*, 299–350.

Gandy, R. (1980). Church's thesis and principles for mechanisms. In J. Barwise, H. Keisler, & K. Kunen (Eds.), *The Kleene Symposium* (pp. 123–148). North-Holland.

Gurevich, Y. (1995). Evolving algebra 1993: Lipari guide. In E. Börger (Ed.), *Specification and Validation Methods* (pp. 9–36). Oxford Univ. Press.

Gurevich, Y. (2000). Sequential abstract state machines capture sequential algorithms. *ACM Trans. on Computational Logic*, *1:2*, 77–111.

Gurevich, Y. (2012). What is an algorithm? In M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, & G. Turán (Eds.), *Sofsem 2012: Theory and Practice of Computer Science* (Vol. 7147, pp. 31–42). Springer. (A slight revision will appear in the proceedings of the 2011 Studia Logica conference on "Church's Thesis: Logic, Mind and Nature.")

Gurevich, Y. (to appear). Semantics-to-syntax analyses of algorithms. In G. Sommaruga & T. Strahm (Eds.), *Turing's ideas — Their Significance and Impact.* Birkäuser/Springer.

Kolmogorov, A. (1953). On the concept of algorithm. *Uspekhi Mat. Nauk*, *8:4*, 175–176 (Russian).

Kolmogorov, A., & Uspensky, V. (1958). On the definition of algorithm. *Uspekhi Mat. Nauk*, *13:4*, 3–28 (Russian). (English translation in AMS Translations 29 (1963), 217–245.)

Shannon, C. (1949). The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, *28*, 59–98.

Turing, A. (1936-37). On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Society*, *42*, 230–265.