What is an Algorithm?

Yuri Gurevich
SOFSEM 2012
The Czech connection of “Gurevich”

- z Hořovice →
- von Horowitz → Horowitz →
- Horowicz, Hurwicz →
- Гуревич →
- Gurevich
Agenda

- Perspective
- Analysis of Turing’s analysis
- On related approaches
- Analysis of sequential algorithms
- Beyond sequential algorithms
Behavioral theory of algorithms

- Syntactically an algorithm is a recipe or program but
- its meaning is a computation that may depend on the environment, e.g. on the initial state.
  - Some logicians disagree.
- We are interested in the meaning, and thus the analysis of algorithms is that of computation.
Computation vs. computability

- Algorithms perform tasks
  - Computing functions is just one class of tasks
  - What function does an operating system compute?

“‘What’s computation’ are more fundamental than ‘what’s computable’.\"
“Give me a fulcrum, and I shall move the world.”

How to analyze computation in general? The task daunting, if not impossible. The notions is amorphous. You need a fulcrum – a viewpoint to make the problem more tractable.

- The quicksand analogy where the desired fulcrum is a rescue board.
Is it possible to define algorithms?

- Not in full generality: the notion keeps expanding.
  - The algorithms/numbers analogy
- But some strata of algorithms have matured. The prime example is sequential (a.k.a. classical) algorithms, the only algorithm from antiquity to 1940s or even 1950s.
Alan Turing
1936/37
Turing’s fulcrum

Only human computers in his time

Forget what’s in the computer’s head; analyze implementation.

Besides, implicitly or explicitly, he constrained the notion of computation.
Some constraints

- Symbolic
- Sequential time
- Bounded step complexity
  - Bounded parallelism
- Isolated, or self-contained.
  - No inter-step or intra-step interference
  - No oracles
  - The initial state determines computation.
Non-symbolic algorithms (satisfying the other constraints)

- Euclid’s
- Ruler-and-compass
- Gauss elimination
- Bisection
Final remarks on Turing’s analysis

- It is hard to isolate first principles.
- How much does the analysis depend on the fact that computers were human?
Functional approaches
von Neumann architecture
Robin Gandy
1980
Gandy’s goal

“Turing's analysis of computation by a human being does not apply directly to mechanical devices.” Hence the goal:

Thesis M. What can be calculated by a machine is computable.
Explicit narrowing

- Sequential time
- The initial state determines the computation.
- "in a loose sense, digital computers."
- Mechanical but not physical (?)
What’s Gandy’s fulcrum?

To the best of my understanding, it is this:

The states can be encoded by hereditary finite sets.

- Principle 1. States are hereditarily finite sets plus ...
- Principles 2 and 3 are technical restriction of the form of states and transitions.
- Principle 4 is about local causality, based on
  - Components can be only so small, by quantum mechanics.
  - Signals move only so fast, by relativity.
Thesis P. A discrete deterministic mechanical device satisfies principles I–IV below.

Theorem. What can be calculated by a device satisfying principles I–IV is computable.
Pros and cons

+ Gandy pioneered axiomatic approach in the analysis of computation.
  
  – What devices satisfy Principle 1?
    ■ There is just one example (cellular automata) and even it is unconvincing.
  
  – We have another (synchronous) parallel computation model which happens to be hard to work with.
Andrey
Kolmogorov
1950s
Facts

“Algorithms compute in steps of bounded complexity”.

- Seq. time
- Bounded step complexity

Kolmogorov published no real analysis of computation.

- 1953 talk summary
- 1958 paper with Uspensky on Kolmogorov machines.
Speculation

Apparently (according to Leonid Levin, a student of Kolmogorov):

Kolmogorov’s fulcrum was to view computation developing in time-space
YET ANOTHER ANALYSIS
Analysis of seq. algorithms

Agenda

Why?
Constraints that seq. algorithms satisfy
Definition of seq. algorithms and Representation Theorem
Derivation of Church’s thesis
The specification problem

By the 1980s, there were plenty of computers and software, and a problem arose how to specify software.

Popular approaches – denotational and algebraic – were declarative.

Declarative specs tend to be cleaner but they are static.

Executable specs: run, test, debug.
Related foundational problem

Maybe executable specs do not have to be low-level and unnecessarily detailed.

Is there an executable spec of any algorithm A on the level of abstraction of A itself?

Maybe some generalization of Turing machines.
Provisos

By default algorithms are sequential.
Informal definition

Kolmogorov 1953: “Algorithms compute in steps of bounded complexity.”

- An algorithm is a deterministic transition system.
- There is a bound, independent from the initial state, on the amount of work performed at each step.
Fulcrum

States are first-order structures.
CONSTRAINTS
Sequential Time

An algorithm is a transition system determined by states, initial states and the transition function $X \rightarrow X'$. 
Interaction?

- No to intra-state interaction.
- Yes to inter-state interaction.
Behavioral equivalence

Two algorithms are behaviorally equivalent if they have same states and transitions.

The equivalence survives inter-step meddling.
Abstract State

The states can be faithfully represented by first-order structures of the same finite vocabulary in such a way that

- state transition does not change the base set,
- collection of states and initial states are closed under isomorphisms, and
- any isomorphism from a state \( X \) to a state \( Y \) is also an isomorphism from \( X' \) to \( Y' \).
Locations and update sets

Locations and their contents

\[ \lambda = (f,(a_1,\ldots,a_j)) \]
\[ \text{Content}(\lambda) = f(a_1,\ldots,a_j) \]

Updates

\[ (\lambda,v) \]

The update set at state \( X \) is

\[ \Delta(X) = \{ (\lambda,v) : v = \text{Content}(\lambda) \text{ in } X' \neq \text{Content}(\lambda) \text{ in } X \} \]
Bonded work

“Algorithms compute in steps of bounded complexity.” Same intuitive idea: there is a bound on the amount of work done during any one step.

But how to measure either?

Fortunately the abstract state constraint helps as we will see.
Reduction to exploration

Without loss of generality, the executor does not remember any history (even its position in the current program.

In order to change state X, the algorithm explores a portion of X and then performs the necessary changes of the values of the predicates and operations of X.

If the exploration is bounded then so is the number of values changed and the change depends only on the explored part.
Bounded Exploration

There is a finite set $T$ of terms such that for all states $X, Y$
if $\text{Val}_X(t) = \text{Val}_Y(t)$ for $t \in T$
then $\Delta(X) = \Delta(Y)$. 
Axiomatic definition of algorithms

View the constraints as postulates.

Definition. A (sequential) algorithm is any object satisfying the three postulates:

- sequential time,
- abstract state,
- bounded-exploration.
Given: A circle $C$ with center $p$, and a point $q$ outside $C$.

The problem:
Construct a tangent at $C$ through $q$

The solution:
Classical non-Turing bisection algorithm

Given a continuous function $f$ with $f(a) < 0 < f(b)$ and given $\varepsilon > 0$, find $m$ with $|f(m)| < \varepsilon$. 

```plaintext
while $|f(m:=(a+b)/2)| \geq \varepsilon$ do
  if $f(m) > 0$
    then $b := m$
  else $a := m$

output := m
```
Representation Theorem

For every seq algorithm A, there exists a seq. ASM behaviorally equivalent to A.
Seq ASM programs

Syntax

\[ f(t_1, \ldots, t_j) := t_0 \]

\[ \text{do in parallel} \]
\[ R_1 \ldots R_k \]

\[ \text{if } t \text{ then } R_1 \text{ else } R_2 \]

Semantics

\[ \Delta = \{(\lambda, a_0)\} \]

where \( \lambda = (f, (a_1, \ldots, a_j)) \)

and each \( a_i = \text{Val}(t_i) \)

\[ \Delta(R_1) \cup \ldots \cup \Delta(R_k) \]

if Val(t) = true then \( \Delta(R_1) \)

else \( \Delta(R_2) \)
THANKS