A Formalization and Proof of the Extended Church-Turing Thesis

Nachum Dershowitz Evgenia Falkovich

One of the important parameters in the description of a modern algorithm is its complexity, or—more precisely—the complexity class it belongs to. To determine the latter, one counts the number of steps required by the algorithm to perform the calculation, relative to input size. That, of course, means that the analysis is sensitive to the chosen measure of the domain elements and to the definition of a single step. To avoid confusion, one is usually required to choose as a measure the size of the (more or less) minimal possible representation of elements. For example, natural numbers are measured by the length of their binary encoding, not their unary one. Then, some well-established theoretical model (such as a Turing machine or RAM) is used to count the number of steps in runs of the algorithm in question over the particular representation. But this makes the definition of complexity model dependent. This dependence remains even if we consider only the asymptotic behavior of the time-complexity function. For example, Kolmogorov-Uspensky machines cannot be simulated in "real time" by a Turing machine [3]. On the other hand, one usually considers complexity classes like **P**, **NP**, or **Exp** to be well defined regardless of the underlying computation model. This invariance relies on the famous *Extended Church-Turing Thesis*, as enunciated, for example in [6], asserting that every "effective" algorithm can be simulated by a Turing machine with only a polynomial overhead in the number of steps. True, this thesis is satisfied by all the usual "effective" models, and makes one optimistic that the hypothesis holds generally. But, to date, there has been no proof precluding the intriguing possibility of the existence of more powerful, but still effectively realizable, computational models than those we have today.

Our goal is to prove the thesis from first principles. To prove it, one should not rely on any specific computational model, but rather capture the generic notion of effective algorithm. We suggest an axiomatic approach, as initiated by Gurevich in [5], for the characterization of classical (sequential) *algorithms*. Furthermore, we focus on their "effectiveness", as axiomatized in [2]. This approach allows us to investigate the notion of effective algorithm on the most basic, yet most intuitive, level.

An outline of our demonstration of the thesis is as follows:

1. We adopt the characterization of (sequential) *algorithms* of Gurevich [5].
2. We consider *implementations*, which are algorithms operating over a specific domain.
3. We adopt the (threefold) characterization of *effective* algorithms in [2].
4. We adopt the formalization of *simulation* under different representations, as described in [2].
5. We represent domain elements by their minimal constructor-based graph (dag) representation; cf. [8].
6. We measure the size of input as the size of this representation.
7. We simulate effective algorithms by abstract state machines [5] in the precise manner of [1].
8. We show that each ASM step is simulatable in a linear number of RAM steps.
9. We show that input states can be properly encoded.

An algorithm, in its classic sense, is a time-sequential state-transition system, where transitions are (partial) functions on states. Each state is self-contained so that the next state is determined by the current state and the algorithm. Following the formalization of algorithms in [5], the required information in states can

be captured using (first-order) logical structures. Furthermore, an algorithm should be abstract, in the sense that it is independent of the choice of representation. So, an algorithm's states should be closed under isomorphism and transitions should respect isomorphisms. Also, an algorithm must possess a finite description. This intuition is captured by the requirement that there exist a finite set of *critical* (ground) terms over the vocabulary of the states of the algorithm such that whenever two states coincide over the critical terms, the changes made by a transition are the same.

An algorithm is a class of *imlementations*, each computing some (partial) function. An inplementation is determined by the choice of representation, which is reflected in a choice of domain and the corresponding choice of initial data and input entries.

A constructor-based definition of effectiveness for arbitrary domains was given in [2]. A constructive implementation must satisfy a few conditions. One is that its domain is isomorphic to the Herbrand Universe over a subsignature, called its *constructors*. Thus, we may identify each domain element with a unique constructor term. Terms are represented by their minimal dag representation (see [8]). Second, input states agree on the values of all terms (over the whole signature) besides the input terms, and all but finitely may terms share the same value, so input states contain only finite information. Any additional operations provided by initial states (which can result in infinitely many equalities between terms) must themselves be programmable by (bootstrapping) constructive implementations, so—in the final analysis—only constructors are applied to values. The complexity measure should, therefore, count steps in terms only of applications of the basic constructors for the domain.

We measure the size of domain elements as the number of nodes in their constructor-based dag representations. All sub-dags are merged to avoid repeated factors [7]. The resulting data structure allows for a RAM to simulate the state of a constructive implementation. Abstract state machines emulate implementations step-by-step [5, 1]. To simulate a transition, this machine performs a fixed number of boolean comparisons and assignments. We show that RAM can simulate each of these actions within a linear, relatively to the state description, number of state transitions. Next we show that state description growth linearly relatively to the initial state and the number of ASM's steps. Thus we conclude that any effective procedure, terminating within T steps can be simulated by RAM terminating within T^2 steps. That completes the proof that the thesis holds for constructive implementations.

Lastly, we generalize the thesis to implementations over arbitrary (countable) domains. As pointed out above, to assign complexity, one must choose a specific representation. In the most general sense, any effective implementation can be step-by-step emulated by a constructive one [2], which in turn, can be simulated (not step-for-step) by a one using only the domain constructors.

References

[1] A. Blass, N. Dershowitz and Y. Gurevich, "Exact exploration and hanging algorithms", *Proceedings of the 19th EACSL Annual Conferences on Computer Science Logic*, Lecture Notes in Computer Science, Springer, 2010.

[2] U. Boker and N. Dershowitz, "Three paths to effectiveness", *Fields of Logic and Computation*, Lecture Notes in Computer Science 6300, Springer, 2010, pp. 135–146.

[3] D. V. Grigoryev, "Kolmogorov algorithms are stronger than Turing machines", *Journal of Soviet Mathematics* 14 (1980) 1445–1450.

[4] Y. Gurevich, "On Kolmogorov machines and related issues", *Bull. EATCS* 35 (1988) 71–82.

[5] Y. Gurevich, "Sequential abstract state machines capture sequential algorithms", *ACM Transactions on Computational Logic* 1 (2000) 77–111.

[6] I. Parberry, "Parallel speedup of sequential machines: A defense of parallel computation thesis", *SIGACT News* 18 (1986) 54–67.

[7] D. Plump, "Term graph rewriting", *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific Publishing, 1999, pp. 3–61.

[8] C. Seshadri, A. Seth, and S. Biswas, "RAM simulation of BGS model of abstract-state-machines", *Fundamenta Informaticae* 77 (2007) 175–185.