Standing on the Shoulders of a Giant
One Persons Experience of Turings Impact
(Summary of the Alan M. Turing Lecture*)

David Harel

The Weizmann Institute of Science, Rehovot, 76100, Israel

A quote attributed to Isaac Newton says "If I have seen a little further it's because I stand on the shoulders of giants". This was indeed stated by Newton, but the general metaphor of a dwarf standing on a giant goes back many, many years earlier. I would recommend the wonderful 1965 book by Robert K. Merton, referred to fondly as OTSOG (on the shoulders of giants) [1] .

Alan M. Turing (1912-1954) can almost be said to be the Mozart of computer science. He died at a young age, under well-known tragic circumstances, leaving an incredibly brilliant, diverse, and versatile legacy. And we are left with the tantalizing question of what would he have achieved had he lived another 30 or 40 years. Turing conceived of the simplest yet most powerful models of computation, but also contributed the design of some of the most complex computers of his time. He proved that there are many things that computers *cannot* do (already in 1936!), but also taught us that there some amazing things that they *can* do. He carried out pioneering work on the idea of mathematical and computational modeling of biology, but also dealt with the question whether computers can be of human-like intelligence. I feel that Alan Turing will become recognized as one of the most important and influential scientists of all time, possibly alongside other giants like Galileo, Newton, and Einstein.

Many computers scientists feel that large parts of their research are rooted in Turing's work and I am one of them. This talk will discuss briefly three of Turing's main lines of work, and the crucial way in which they impacted one person's research — my own. I will not be discussing a central part of my work, that related to software and systems engineering and executable visual languages, but the talk will cover several of the other topics I have worked in over the years.

Computability

As is well known, Turing was a pioneering force behind the notion of computability and non-solvability [2], and thus, alongside the likes of Church, Gödel, Post

* Versions of this talk were presented in a number of conferences and symposia during 2012, to celebrate the Alan M. Turing Centennial year. Parts of the more recent research reported upon here were supported by an Advanced Research Grant to DH from the European Research Council (ERC) under the European Community's FP7 Programme.

and Kleene, was a central figure in establishing the limits of computing. In particular, Turing showed the halting problem to be undecidable, and essentially invented the notion of universal computing, via the universal Turing machine. At this point, one should definitely mention Rice's theorem [3], which can be viewed as a grand extension of Turing's undecidability result for the halting problem, and which establishes the dramatic fact that no non-trivial problem about computation can be decidable! This includes correctness, equivalence, efficiency, and many others... Thus, as a profession, computing constitutes the ultimate example of the barefoot shoemaker aphorism. See [4] for a detailed exposition of these and other results on the limitations of computing.

My own humble extensions of Turing's work in this area include three topics, which will be discussed briefly below: computability on finite structures, computability on infinite recursive structures, and high non-solvability/undecidability.

In a joint 1979 STOC paper with Ashok Chandra, we addressed the problem of defining the computable functions over finite structures [5]. The issue is the following. Say you want to compute a function on graphs, or on relational databases. These are really sets of tuples (a graph is a set of pairs of vertices), and there is no order on their elements. A function that takes advantage, so to speak, of some ordering that could be the result of a particular representation of the graph (e.g., on a Turing machine tape), should be outlawed. So what is the appropriate notion of computability over such structures? How should one extend Church-Turing computability from words or numbers to general (unordered, or partially unordered) structures? The answer offered in our 1979 paper is that a computable function over a structure has to be partial recursive, in the classical sense of Turing and co., and in addition has to be *consistent* (or *generic*, as the notion was later called), which means that the function has to preserve isomorphisms. Genericity captures the idea that the function should not use any information that is not present in the structure itself, such as an ordering on the tuples.

This definition, however, would have been worthless unless accompanied with a complete language for the computable functions — a sort of analogue for structures of Turing machines or the lambda calculus, or, for that matter, of any programming language over numbers or words. This, of course, raises the question of why not simply take as the complete language the set of generic Turing machines, i.e., exactly all those that preserve isomorphisms? The answer is that by Rice's theorem even the syntax of such a language in non-effective/non-computable: you cannot tell whether a string of symbols is a legal program in the language, because it is undecidable to tell whether a function preserves isomorphisms. What Chandra and I did in our paper [5] was to define a simple query language QL over relations (which is really a variant of the first order relational algebra enhanced by a *while-do* looping construct), and prove it complete. The subtle part of the proof was to show how, given an input relational structure S, we are able to use QL to program a relation that represents the set of automorphisms of S.

The paper itself (which appeared as a journal version in 1980) was written in the setting of relational databases, but it can be viewed as establishing computability over general finite relational structures. Indeed, its main result has been extended over the years to much more complex structures. We believe that this is a good example of the standing-on-the-shoulders-of-the-giant phenomenon, consisting of a natural and modest, yet basic, extension to general structures of Turing's original notion of computability .

A side remark worth including here concerns the second paper written with Chandra, which appeared in the 1980 FOCS conference [6]. There we continued the work on computable queries/functions on (unordered) structures, and defined the structural and the complexity-theoretic basics over such structures. These have been shown to be underlie many issues in descriptive complexity and finite model theory. In that paper, we posed the question of "does QPTIME have an effective enumeration?" (see p. 118 of the 1982 journal version). It can be viewed as asking whether there can be any effective language/logic capturing the polynomial time functions over general unordered structures, such as graphs. This problem (which was popularized in later writings of Gurevich and others) has been open now for over 30 years.

Many years later, with my PhD student Tirza Hirst [7], we extended the notion of computable functions on structures to deal with infinite recursive structures; e.g., relations whose set of tuples is computable. For example, a recursive graph is one whose vertex and edge sets are effective; the simplest case is a recursive binary relation over the natural numbers. We were able to obtain several results on recursive structures, including a completeness result for an appropriate variant of the QL language of [5] in the style of the proof given there [7,8]. This has to be done on a suitably restricted class of recursive structures, because the general class is not closed even under projection.

The third topic that interested me for many years, and which can also be viewed as directly extending the work of Turing (and in this case, that of Stephen Kleene too), is *high undecidability* or high unsolvability. Of specific interest are problems that can be shown to be complete for the lowest level of the analytic hierarchy, the so-called Σ_1^1/Π_1^1 level. Such problems are thus infinitely many levels of undecidability worse than the halting problem.

Here are some of the results I was able to obtain. First, the halting problem of Turing is extended to the halting problem for programs with countably infinite nondeterminism, and also the halting problem for parallel and concurrent programs under the assumption of fairness — that is, fair halting. Actually, these two problems turn out to be closely related, and both were shown to be highly undecidable [9].

Second, with various co-authors and over a period of several years, I was able to show that many satisfiability/validity problems for logics of programs are also highly undecidable, including variants of non-regular propositional dynamic logic, temporal logical in two dimensions, and more [10,11]. Many of these satisfiability results are proved by a reduction from another problem I was able to show highly undecidable — a recurring version of the tiling problem originating with Wang, in which there is an additional requirement that a specific tile has to occur infinitely often in a tiling of the plane [10].

Finally, I was able to show that asking whether a recursive graph has a Hamiltonian path is also highly undecidable, i.e., complete for the Σ_1^1/Π_1^1 level [12]. This problem was known to be undecidable but had not given rise to an upper bound residing in the arithmetical hierarchy. The proof in [12], establishing that the problem is actually outside the arithmetical hierarchy, is a rather picturesque reduction from the halting problem with countable non-determinism (which is essentially the unfoundedness of recursive trees).

In a second paper with Hirst [13], we were able to link high undecidability of properties over recursive structures to the approximability of finitary versions of those problems on the NP level.

Biological Modeling

As is well known, Turing pioneered the mathematical basis for modeling biological growth, or in more technical terms, *morphogenesis* [14]. He was thus one of the first to consider computational and mathematical means for modeling biological processes, specifically pattern-formation and growth.

My modest follow-up to that work involves modeling the development of the pancreas, carried out with Yaki Setty and Yaron Cohen. We used Statecharts and other computer science and software engineering techniques to build a dynamic executable model of a cell, which is "scheduled" to become a pancreatic cell [15,16]. When thousands of such cell statecharts are simulated together, the result is an interactive model that mimics the growth process (organogenesis) of the pancreas, ultimately obtaining its cauliflower- or broccoli-shaped final form. See also [17].

In more recent ongoing work, carried out with Naama Bloch, and also building on Turing's morphogenesis work, we are in the process of modeling the growth of a cancerous tumor. This is done in a manner similar to that used in the pancreas model, and again we model the dynamics of a cancer cell (and the surrounding blood vessels), trying to capture the crux of biological growth.

In the context of our use of Statecharts to model biological growth, the following quote from Turing's 1952 paper, 50 years earlier, is particularly illuminating: "... one proceeds as with a physical theory and defines an entity called 'the state of the system'. One then describes how that state is to be determined from the state at a moment very shortly before" [14].

The Turing Test

As is also very well known, Turing spent a lot of energy in his later years thinking about whether computers can think... The most famous outcome of this process involves Turing's imitation game, which has come to be called the *Turing test*, for determining whether a computer or a piece of software is intelligent [18].

The test involves a human sitting is one room and a computer in another, and a human interrogator in a third room who tries to distinguish one from the other. the computer passes the test if the interrogator is not able to tell the difference. An enormous amount of material has been written about this test and its significance, so there is no reason to recall any of it here.

My modest follow-up, once again as the dwarf standing on the shoulders of the giant, is a proposal for a Turing-like test for modeling nature. Continuing the subject matter of the previous section, one of the things I've been talking about for many years is a grand challenge in the area of systems biology for comprehensive and realistic modeling. The challenge is to construct a full, correct, true-to-all-known-facts, four-dimensional model of a multi-cellular organism. In short, the challenge is to build an interactive executable model of an animal [19]. As a second-level part of this proposal, I suggested using the *C. elegans* nematode as the model organism for tackling the challenge.

However, the question arises as to when you know that such a model is complete, so that you can satisfy yourself with its validity. This is to be contrasted with conventional modeling and analysis work in bioinformatics or systems biology, where you start out with a question or a set of questions, in which case the model is complete and valid when it can be shown to provide answers in full accordance with the answers that one gets in the laboratory. One can then go on to the next project and the next set of questions. In contrast, modeling an entire biological system — an organism such as a worm, a fly, or an elephant, or even just a complete organ such as a pancreas, a liver, a heart or a brain — the question arises as to when the project ends and the model can be claimed to be valid.

My 2005 suggestion for addressing this question is a Turing-like test, but with a Popperian twist [20]. In line with Turing's original test, one places the computer with its model in one room, and in the other one sets up an advanced laboratory researching the actual organism being modeled. If we are to take the *C. elegans* worm as the modeled organism, we should have an advanced *C. elegans* laboratory in the second room. Then, a team of interrogators, well-versed in the subject matter, is given time to probe the laboratory and the model, trying to tell the difference. Of course, the buffering between the interrogators and the two rooms has to be more elaborate than that in Turing's original test for intelligence, since, for example, many questions can be answered in a split second by a computer but could take months (or could be actually impossible) to answer in the laboratory. But this is a technicality I will not get into here.

The interesting twist present in this new version of the Turing test is that a model passing it can indeed be claimed to be a true model of an elephant, a *C. elegans*, or a liver, yet such a conclusion should be taken to be temporary, and should be stamped on the computer in non-permanent ink. The reason is that, as Carl Popper has taught us, this kind of definitive statement will no doubt be refuted in the future when more is discovered about the organism or the organ being modeled. Hence, such a model is really a theory of the biological system being modeled, and the theory will have to be revised when new facts are discovered, which is actually exactly what we want! See the discussion in [20].

* * *

In conclusion, all these are but very personal and idiosyncratic examples of the work of one particular computer scientist over a period of almost 35 years. They can all be viewed as modest extensions and generalizations of Turing's pioneering work in three different areas.

The emerging picture is definitely that of a dwarf standing on the shoulders of a true giant, and as a result perhaps being able to see just a tiny bit further.

References

1. Merton, R.K.: On The Shoulders of Giants: A Shandean Postscript. Free Press (1965)
2. Turing, A.M.: On Computable Numbers with an Application to the Entscheidungsproblem. Proc. London Math. Soc. 42, 230–265 (1936); Corrections appeared in: ibid 43, 544–546 (1937)
3. Rice, H.G.: Classes of recursively enumerable sets and their decision problems. Trans. Amer. Math. Soc. 74, 358–366 (1953)
4. Harel, D.: Computers Ltd.: What They Really Can't Do. Oxford University Press (2000); Revised paperback edition, 2003. Special Turing Centennial printing (2012)
5. Chandra, A.K., Harel, D.: Computable Queries for Relational Data Bases. J. Comput. System Sciences 21, 156–178 (1980); Also, Proc. ACM 11th Symp. on Theory of Computing, Atlanta, Georgia, pp. 309–318 (April 1979)
6. Chandra, A.K., Harel, D.: Structure and Complexity of Relational Queries. J. Comput. System Sci. 25, 99–128 (1982); Also, Proc. 21st IEEE Symp. on Foundations of Computer Science, Syracuse, New York (October 1980)
7. Hirst, T., Harel, D.: Completeness Results for Recursive Databases. J. Comput. System Sci. 52(3), 522–536 (1996); Also, Proc. 12th ACM Symp. on Principles of Database Systems, pp. 244–252. ACM Press, New York (1993)
8. Harel, D.: Towards a Theory of Recursive Structures. In: Enjalbert, P., Mayr, E.W., Wagner, K.W. (eds.) STACS 1994. LNCS, vol. 775, pp. 633–645. Springer, Heidelberg (1994)
9. Harel, D.: Effective Transformations on Infinite Trees, with Applications to High Undecidability, Dominoes and Fairness. J. Assoc. Comput. Mach. 33, 224–248 (1986)
10. Harel, D.: Recurring Dominoes: Making the Highly Undecidable Highly Understandable. Ann. Disc. Math. 24, 51–72 (1985); Also, Karpinski, M. (ed.) FCT 1983. LNCS, vol. 158, pp. 177–194. Springer, Heidelberg (1983)
11. Harel, D.: How Hard is it to Reason About Propositional Programs? In: Broy, M. (ed.) Program Design Calculi. NATO ASI Series, vol. F-118, pp. 165–184. Springer, Berlin (1993)
12. Harel, D.: Hamiltonian Paths in Infinite Graphs. Israel J. Math. 76(3), 317–336 (1991); Also, Proc. 23rd ACM Symp. on Theory of Computing, New Orleans, pp. 220–229 (1991)
13. Hirst, T., Harel, D.: Taking it to the Limit: On Infinite Variants of NP-Complete Problems. J. Comput. System Sci. 53(2), 180–193 (1996); Also, Proc. 8th IEEE Structure in Complexity Theory, pp. 292–304. IEEE Press, New York (1993)

14. Turing, A.M.: Computing Machinery and Intelligence. Mind 59, 433–460 (1950)
15. Setty, Y., Cohen, I.R., Dor, Y., Harel, D.: Four-Dimensional Realistic Modeling of Pancreatic Organogenesis. Proc. Natl. Acad. Sci. 105(51), 20374–20379 (2008)
16. Setty, Y., Cohen, I.R., Harel, D.: Executable Modeling of Morphogenesis: A Turing-Inspired Approach. Fundamenta Informaticae (in press)
17. Fisher, J., Harel, D., Henzinger, T.A.: Biology as Reactivity. Comm. Assoc. Comput. Mach. 54(10), 72–82 (2011)
18. Turing, A.M.: The chemical basis of morphogenesis. Philosophical Transactions of the Royal Society of London B 327, 37–72 (1952)
19. Harel, D.: A Grand Challenge for Computing: Full Reactive Modeling of a Multi-Cellular Animal. Bulletin of the EATCS, European Association for Theoretical Computer Science (81), 226–235 (2003); Early version prepared for the UK Workshop on Grand Challenges in Computing Research (November 2002), Reprinted in: Paun, Rozenberg, Salomaa (eds.) Current Trends in Theoretical Computer Science: The Challenge of the New Century, Algorithms and Complexity, vol. I, pp. 559–568. World Scientific (2004)
20. Harel, D.: A Turing-Like Test for Biological Modeling. Nature Biotechnology 23, 495–496 (2005)