Will I be Pretty, Will I be Rich?

Some Thoughts on Theory vs. Practice in Systems Engineering

(Short Summary of Talk)

David Harel
The Weizmann Institute of Science, Rehovot, Israel
harel@wisdom.weizmann.ac.IL

"The mathematician's patterns, like the painter's or the poets's, must be *beautiful*; ... there is no permanent place in the world for ugly mathematics."
(G. H. Hardy [H, p. 25])

Person A: "I'm writing a best-seller."
Person B: "Short of money, hah?"
(Cartoon in the *New Yorker*)

1 Preamble

This is a very short summary of a talk presented at the 13th ACM Symposium on Principles of Database Systems (PODS) in Minneapolis in May, 1994. The talk attempted to put forward some thoughts on theoretical vs. applied research in system design and programming. By its very nature, such a talk is bound to be disorganized, rambling, non-self-contained, and extremely subjective. It was; and the written summary you are reading is even worse, since it not only omits the details of the examples used in the talk, but also lacks the intonations, facial gestures and hand-waving that are part and parcel of talks that have little technical content.

Oh well. So be it.

2 A 3-way Classification

This is a "Principles of" conference. Most of its participants do theory — that is, research whose methods and tools are mathematical — but theory geared towards particular kinds of real systems. In this case, it is database systems. In other analogous conferences, such as those on Principles of Programming Languages (POPL) or Principles of Distributed Computing (PODC), the setup is the same, but the kinds of systems differ.

What sort of theory do we do here, and why? Should the general theory community take notice of us? Should the applied crowd listen in? How about the converse questions: Should we peddle our merchandise to other theoreticians? Are we doing enough to serve the needs of our own real-world practitioners?[1]

At the heart of the talk was an attempt to clarify some of the issues behind these questions, by dividing the research carried out by theoreticians into three kinds, which will be referred to as Type 1, Type 2 and Type 3 theory.[2]

Type 1 theory concerns true foundations and principles. It should be robust, deep and of fundamental nature, and should explain, generalize and enlighten. Such are the basics of computability and complexity theory, for example, as they emerge from the work of Turing, Church, Cook and many others.

Type 2 theory responds directly to the needs arising in applications. It should be pragmatic and specific, molding itself to fit the requirements posed by real-world difficulties, and it should result in things that work and can actually be used. Such is the Fast Fourier Transform, for example, or those parts of the theory of context-free languages that lead to efficient compilation techniques.

Type 3 is theory for the sake of theory (TST). It should be mathematically elegant, yet difficult and clever, and should be of interest to other theoreticians. Much of the work we do is of this type.

The borderlines between these are fuzzy, and as time goes by migration often take place: Many Type 3 results and techniques eventually become Type 1, and

[1] In the context of the present talk, these would include many of the non-PODS members of the SIGMOD community.

[2] While our interest here is mainly in theory carried out in conjunction with practical fields of computer science, such as databases and systems engineering, many of the points made can be modified to apply to theory in general. Also, the 3-way classification proposed here is somewhat different from the one proposed by Raghavan [R] for general STOC/FOCS theory.

SIGMOD/PODS 94 - 5/94 Minneapolis, Minnesota USA
© 1994 ACM 0-89791-639-5/94/0005..$3.50

sometimes — but more rarely — Type 3 work becomes applicable, converting it into Type 2.

TST is legitimate and desirable, and not only because it might get upgraded. It is absolutely essential to the well-being and substance of a scientific community. Even so, most theoreticians will never admit to doing Type 3 work.[3]

One difference between Types 1 and 2 on the one hand and Type 3 on the other is in the judges. While the quality of TST is inevitably determined by theoreticians, the ultimate test of both Type 1 and Type 2 is in the opinions of real-world people, such as systems engineers and programmers. A non-applicable piece of work can be considered by theoreticians to be Type 1, but it cannot fully deserve that label unless engineers and programmers can be made to appreciate its virtues too. Otherwise, there are exactly two possibilities: (i) the theory is bad, or (ii) it is TST (in which case it might be excellent, but the applied guys couldn't really have known).

3 Did Codd do Theory?

This part of the talk was dedicated to illustrating the points with examples taken from database and system engineering research.

Here are some of the questions one can ask in connection with a particular piece of work: Are Codd's early papers on relational databases and query languages [C1, C2] PODS-like material? Was his work on defining the relational model Type 1, 2 or 3? How about his definition of the relational calculus and algebra and the proof of their equivalence?

Most of the examples given in the talk were couched as contrasting pairs, in an attempt to identify the totally different attitudes and strengths a theoretician must summon up in order to carry out Type 1 vs. Type 2 work. Besides Codd's pioneering work, the database topics discussed included Query-by-Example [Z], Datalog (cf. [U]), computable queries [CH1], and fixpoint and *while* queries [AU, AV, CH2]. Also mentioned was Fagin's result concerning NP and existential second order logic [F], which inspired many of the connections between computational and descriptive complexity (e.g., those in [I, V, AV]).

To further illustrate things, several examples were given from the world of system development. Specifically, Petri nets [Re] and statecharts [Ha1] were discussed from a user's point of view, as were results on the relative expressive power and succinctness of these and other formalisms for specifying system behavior [RS, MF, EZ, DH]. Also illustrated was the contrast between the theoretical and practical aspects of verifying finite-state systems by executable specifications (see, e.g., [Ha2]), or the recently proposed methods based on BDD's [B, B+].

[3]Hardy, the great number theorist, was a notable exception, stating, in the famous passage from [H, p. 90]: "I have never done anything 'useful'. No discovery of mine has made, or is likely to make, directly or indirectly, for good or ill, the least difference to the amenity of the world. [...] I have just one chance of escaping a verdict of complete triviality, that I may be judged to have created something worth creating." He was wrong, of course, as any modern-day cryptographer will tell you.

4 Post-Ramble

There was also a message in all this. Subjective, and perhaps trivial, but here it is anyway.

A typical theoretician wants his or her work to end up being Type 1. However, setting out in advance with this in mind is usually pointless. We can try to *aim* in the direction of Type 1 by being collective and general. We should avoid overly specialized theories, ones that seem to apply only to a special case of some special language, model or approach. We should seek results that are as generic and as all-encompassing as possible. Robustness is the name of the game. And we should always keep in mind that the essence of true Type 1 must be appreciable by non-theoreticians too, and it is our responsibility to expose and elucidate it.

As to Type 2, while theory people are by no means obliged to produce applicable work, some of us really want to. If we are interested in *actively* carrying out Type 2 work, we should get out there and become involved. We should take a real interest, listen attentively to what the real-world people ask for, and study their thought-patterns and work-habits. Only then can we try to see if there are ways we can help. The problems arising out there are usually much harder than we tend to think. Riches don't come easy. Doing our work in isolation, and then trying to impose our ideas on the real world, is bound to fail. If engineers and programmers do not find it beneficial to use the result of an application-oriented research effort — for whatever reasons — that piece of research is probably quite useless. We should be humble; they are the absolute judges.

So much for us theoreticians. What can be said here to the practitioners?

Well, as far as Type 2 theory goes, simply don't give in. Be demanding; be pedantic, or even idiosyncratic. Explain and justify your problems and needs to the theoreticians. Let them in on your whims and fancies; you might just turn lucky. But be patient, since most theoreticians cannot muster the down-to-earth attitude an engineer needs in order to function well in the face of real-world problems. Some of us can't even program!

When it comes to Type 1 work, the practitioners should be the ones to show an interest. Theory can be more than just pretty mathematics. Some of it is deep, sweeping and fundamental. It will usually not be of direct help in your daily work, but it very often addresses truly basic issues, capturing phenomena that are at the heart of the field — that field in which your real-world work is done. Be open. Listen to it. It might not be quite as way-out as you think.

References

[AU] A.V. Aho and J.D. Ullman, "Universality of Data Retrieval Languages", *Proc. 6th ACM Symp. on Principles of Prog. Lang.*, 1979, pp. 110–117.

[AV] S. Abiteboul and V. Vianu, "Generic Computation and its Complexity", *Proc. 23rd ACM Symp. on Theory of Computing*, 1991, pp. 209–219.

[B] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Computers* C-**35**:8 (1986), 677–691.

[B+] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill and J. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond", *Inf. and Comput.* **98** (1992), 142–170.

[CH1] A.K. Chandra and D. Harel, "Computable Queries for Relational Data Bases", *J. Comput. Syst. Sci.* **21** (1980), 156–178.

[CH2] A.K. Chandra and D. Harel, "Structure and Complexity of Relational Queries", *J. Comput. Syst. Sci.* **25** (1982), 99–128.

[C1] E.F. Codd, "A Relational Model of Data for Large Shared Data Banks", *Comm. Assoc. Comput. Mach.* **13**:6 (1970), 377–387.

[C2] E.F. Codd, "Relational Completeness of Data Base Sublanguages", In *Data Base Systems* (Rustin, ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972.

[F] R. Fagin, "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets", In *Complexity of Computations* (R. Karp, ed.), SIAM-AMS Proceedings, Vol. 7, 1974, pp. 43–73.

[DH] D. Drusinsky and D. Harel, "On the Power of Bounded Concurrency I: Finite Automata", *J. Assoc. Comput. Mach.*, in press. (Preliminary version appeared in *Proc. Concurrency '88*, LNCS 335, Springer-Verlag, New York, 1988, pp. 74–103.)

[EZ] A. Ehrenfeucht and P. Zeiger, "Complexity Measures for Regular Expressions", *J. Comput. Sys. Sci.* **12** (1976), 134–146.

[H] G.H. Hardy, *A Mathematician's Apology*, Cambridge Univ. Press, 1940.

[Ha1] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", *Sci. Comput. Prog.* **8** (1987), 231–274.

[Ha2] D. Harel, "Biting the Silver Bullet: Toward a Brighter Future for System Development", *Computer* (Jan. 1992), 8–20.

[I] N. Immerman, "Relational Queries Computable in Polynomial Time", *Inf. and Cont.* **68** (1986), 86–104.

[MF] A.R. Meyer and M.J. Fischer, "Economy of Description by Automata, Grammars, and Formal Systems", *Proc. 12th IEEE Symp. on Switching and Automata Theory*, 1971, pp. 188–191.

[RS] M.O. Rabin and D. Scott, "Finite Automata and Their Decision Problems", *IBM J. Res.* **3** (1959), 115–125.

[R] P. Raghavan, Electronic mail contribution to a debate on the future of theory, Feb. 17, 1994.

[Re] W. Reisig, *Petri Nets: An Introduction*, Springer-Verlag, Berlin, 1985.

[U] J.D. Ullman, *Principles of Database and Knowledge-Base Systems*, Vols. I and II, Computer Science Press, 1988.

[V] M. Vardi, "The Complexity of Relational Query Languages", *Proc. 14th ACM Symp. on Theory of Computing*, 1982, pp. 137–146.

[Z] M.M. Zloof, "Query-by-Example: A Data Base Language", *IBM Systems J.* **16** (1977), 324–343.