Review of International Geographical Education | RIGEO | 2020

RIGEO

ISSN: 2146 - 0353

Review of International
GEOGRAPHICAL EDUCATION

www.rigeo.org

COMPUTER SCIENCE AND MATHEMATICAL LOGIC

Dr N Prabhudeva[1], A M Girija[2], Jayashree D N[3]

Professor & HOD[1], Asst. Professor[2], Associate. Professor[3],

drnprabhudeva@gmail.com[1] , girijaam09@gmail.com[2] , jayashreedn12@gmail.com[3]

Department of Mathematics, Proudhadevaraya Institute of Technology, T.B. Dam Road, Hosapete,Karnataka-583225

1. Introduction

Computer science has strong connections with numerous aspects of mathematical logic, but those aspects are sometimes different from those traditionally studied for pure mathematical purposes. This paper is a survey of some of those interactions. Its main themes are (1) logic's clarification of computational concepts and (2) computation's infusion of new concepts and questions into logic.

Section 2 mentions some of the earliest interactions. They involve what would now be called computability theory or recursion theory, rather than computer science, as they do not involve any resource limitations. Nevertheless, I consider them worth pointing out, to emphasize the contribution of mathematical logic to a broad topic arising in diverse fields of mathematics.

Section 3 concerns first-order logic, especially the notion of structure at the basis of the semantics of first-order logic, pointing out that this same notion of structure is well-suited for many roles in computer science. I also describe here a computational view of the expressive power of first-order logic.

Section 4 concerns logical systems obtained by adding to first-order logic the central non-first-order concept involved in computation, namely the notion of iteration. The logical incarnation of iteration is

given by fixed-point operators, which are involved in the logical characterizations of various complexity classes. Fixed-point operators play a role in all the subsequent sections of this paper.

Section 5 begins with a very general conjecture of Yuri Gurevich that polynomial-time computation, on general finite structures, cannot be exactly captured by any logic (in a very broad sense of "logic"). I include a description of a logic, "choiceless polynomial time," that comes surprisingly close to capturing PTime, and I discuss the possibility that it might actually be a counterexample to Gurevich's conjecture.

Section 6 is devoted to a logic of a rather different sort, obtained from first-order logic by not only adding a fixed-point operator but also removing part of the machinery of first-order logic, namely the universal quantifier. This existential fixed-point logic turns out to have strong connections to computation as well as a certain mathematical elegance.

Fixed-point logics do not admit a complete deductive system in the usual sense, but it is not difficult to write down a reasonable system of axioms and rules governing the behavior of a fixed-point operator. The question then arises whether the unavoidable incompleteness of such a system affects only complicated, Gödel-style sentences or whether very simple truths might be unprovable. In Section 7, I describe the system that I have in mind and conclude with an open question about a simple truth that might be unprovable.

2. Prehistory

In the early part of the twentieth century, several branches of mathematics had encountered fundamental questions about algorithmic issues. Examples include the following.

Number theory: Given a polynomial equation in many variables, with integer coefficients, determine whether there is an integer solution. (Hilbert's tenth problem [29])

Topology: Given two topological manifolds, in the form of finite triangulations, determine whether they are homeomorphic.

Group theory: Given a group, by means of (finitely many) generators and relations, and given two words in the generators, determine whether these words represent the same element of the group. (The Word Problem [24])

Logic: Given a first-order sentence, determine whether it is logically valid. (The Decision Problem)

In each of these cases, the problem was originally to find an algorithm. For example, Hilbert's statement of his tenth problem is, after specifying the equations under discussion, "One is to give a procedure by which it can be determined by a finite number of operations whether the equation is solvable in rational integers."[1] Similarly, Dehn's statement of the word problem is, "One is to give a method to decide, with a finite number of steps, whether [a given] element is equal to the identity or not."[2] And Hilbert and Ackermann [30] stated the decision problem as, "The decision problem is solved if one knows a procedure which, given a logical expression, allows one to decide by finitely many operations its validity resp. its satisfiability."[3]

Logicians provided a completely new way of viewing such problems. They replaced "Find an algorithm ..." with "Is there an algorithm ...?" That is, they introduced the notion that there might not exist an algorithm and (crucially) that such a result might be provable. This means that, instead of the then prevalent notion of algorithm, namely "I'll know it when I see it," one needs a precise mathematical notion of algorithm, or at least of algorithmic computability. Such definitions were provided independently by Church [16] using the λ-calculus and by Turing [38] using what are now known as Turing machines. Undecidability results, i.e., theorems asserting the nonexistence of algorithms for various problems, followed. The first was Turing's proof of the undecidability of the decision problem for first-order logic.

For all four of the problems listed above, algorithms were known for some special cases, but the general cases turned out to be unsolvable.

- The existence of solutions to *linear* systems of Diophantine equations is decidable by algorithms known since antiquity, but once one progresses even to quadratic equations, it becomes necessary to limit either the number of equations in the system or the number of variables. The general case was proved undecidable by Matijasevič, adding the final step to work of Davis, Putnam, and Robinson; for more information, see [21] and the references there. The general case can be reduced to the quadratic case by using more variables and more equations; furthermore, any number of quadratic equations can be reduced to a single quartic equation.
- The homeomorphism problem admits an algorithmic solution for 2-dimensional manifolds; it suffices to check the Euler characteristic and orientability (and, in the case of manifolds with boundary, the number of boundary components), which can easily be done combinatorially. In the case of 4-dimensional manifolds, the homeomorphism problem is known to be algorithmically unsolvable, essentially as a consequence of unsolvability of the isomorphism problem for groups. The intermediate case of 3-manifolds is apparently slightly open. There is an algorithm in Chapter 1 of [4] which uses many high-powered results, including Perelman's proof of Thurston's geometrization conjecture, and which was thought to solve the homeomorphism problem

for 3-manifolds. But recently a gap[4] has been found and apparently not yet filled, so this case of the homeomorphism problem is still open.

- The word problem for groups was solved by Dehn [24] in the special case that each generator occurs at most twice in all the relations together. Algorithmic solutions are now known for far broader classes of group presentations; see for example [12] and the references of the papers there.
- The decision problem for first-order logic had been proved decidable for some special classes of sentences, when Church [16] and Turing [38] showed that the general case is undecidable. For a thorough discussion of decidability and undecidability results for sentences with special structure, see [13].

The situation can be summarized as follows: The desire for algorithmic solutions of various problems pervaded much of mathematics. Only one area, mathematical logic, produced (1) the idea that algorithmic computability could be defined with sufficient precision and generality to admit mathematical analysis and perhaps proofs of unsolvability, (2) specific convincing proposals for such precise definitions (especially Turing's analysis in [38]), and (3) the first actual undecidability proofs.

Although this work now falls under the heading of computability theory rather than computer science, and although it was done before digital computers were built, it has a strong influence on modern computer science. For example, not only are Turing machines used as the basis for common definitions of resourcebounded complexity classes, but the diagonalization method, used by Turing to prove undecidability, became one of the basic tools in the study of these classes. The notion of complexity hierarchies, which is of wide use in theoretical computer science, also traces its origin to the arithmetical hierarchy and related structures, such as the Turing degrees, in computability theory.

3. First-order logic

The concept of a first-order *structure*, i.e., a set with specified finitary relations and functions on it (and distinguished elements, but I view those as 0-ary functions), can be used to represent a wide variety of important entities from computer science (as well as other areas). For example, all of the following can be regarded as first-order structures.

Databases: The relational database model of [17] amounts to a first-order structure with only relations and constants, not functions of arity ≥ 1.

Words over an alphabet: A word w over an alphabet A is usually modeled as a finite linearly ordered set (the set of locations in the word) with a unary predicate U_a for each letter $a \in A$. The locations that satisfy U_a are those that are occupied by the letter a in w.

Linked lists: A list can be modeled by a structure whose elements are locations in the list. There are relations (as in the case of words above) or a function describing the content of each location, there is another function assigning to each location the next one in the list (and assigning to the last element itself), and there is a constant for the first element of the list.

Other data structures: Examples include trees and arrays.

Configurations of Turing machines: These configurations amount to words with additional constants for the scanned square and the control state.

Configurations of other hardware: See, for example, [37] and other papers under the "hardware" link at [1].

In his general analysis of sequential computation [27] Gurevich proposed that "any kind of static mathematical reality can be faithfully represented as a first-order structure," supporting this proposal by alluding to "the huge experience of mathematical logic". This proposal is at the basis of the development of abstract state machines (ASMs), and the continuing applications of ASMs in diverse areas of computing have not produced reasons for doubting the proposal.

One might add that the restriction to static reality is not strictly necessary, because nothing prevents a first-order structure from including a time coordinate. For example, first-order structures can model not only instantaneous configurations but entire computations of Turing machines. Such structures are an essential ingredient in the standard proof (as in [19] or [25, Section 2.6]) of the NP-completeness of propositional satisfiability. The reason for "static" in the quotation from [27] is that ASMs model dynamics in another way.

It is not only the notion of first-order structure but also the notion of first-order formula that is relevant to computing. First-order formulas can serve to describe single steps in parallel computation. More precisely, if the state, at any moment, of a computation is modeled by a first-order structure, and if individual computation steps are allowed to contain much parallel work but only bounded sequentiality, then the effect of a step on the state can be described by a first-order interpretation. Conversely, the truth value of a first-order formula can be evaluated by such a step. For a general discussion of this idea, see [5], and for details see [9].

4. Iteration and fixed points

Repetition of simple, first-order steps leads beyond first-order logic. For example, if a structure consists of a set with a binary relation, then the transitive closure of that relation is ordinarily not first-order definable in the structure. For some purposes, it is useful to extend first-order logic with a transitive-closure construct, allowing formulas to refer to the transitive closure of any definable relation. More often, though, one needs a more powerful construct, a fixed-point operator, which can be defined as follows.

Suppose $\gamma(P, x)$ is a formula containing the k-ary predicate variable P and a k-tuple of (first-order) variables $x = (x_1, \ldots, x_k)$. It defines, in any structure A (whose underlying set we denote, as usual, by the corresponding italic letter A), an operator Γ on k-ary relations by

$$\Gamma(P) = \{\vec{a} \in A^k : \mathfrak{A} \models \gamma(P, \vec{a})\}.$$

If this operator is monotone, i.e., if $P \subseteq Q$ always implies $\Gamma(P) \subseteq \Gamma(Q)$, then iterating it produces the *least fixed-point* Γ^∞. In more detail, one defines, by transfinite induction on ordinals,

$$\Gamma^0 = \emptyset, \qquad \Gamma^{\alpha+1} = \Gamma(\Gamma^\alpha), \qquad \text{and} \qquad \Gamma^\lambda = \bigcup_{\alpha<\lambda} \Gamma^\alpha \text{ for limit } \lambda.$$

The sequence of sets Γ^α is monotone (with respect to $\subseteq$) and therefore eventually stabilizes. The stable value, Γ^α for all sufficiently large α, is defined to be Γ^∞. It is easy to prove that Γ^∞ deserves the name of least fixed-point; we have $\Gamma(\Gamma^\infty) = \Gamma^\infty$, and if any other P satisfies $\Gamma(P) = P$ (or even only $\Gamma(P) \subseteq P$) then $\Gamma^\infty \subseteq P$.

If Γ is not monotone but inflationary, i.e., $\Gamma(P) \supseteq P$ for all P, then the same iteration produces a fixed-point Γ^∞, but it need not be the smallest fixed-point of Γ. It is called the *inflationary fixed-point* [22].

The least fixed-point of a monotone operator can be obtained not only by the standard iterative process described above but by any iteration that adds, at each step, to the current P, some nonempty subset of $\Gamma(P) - P$. That is, one can add, at each step, some but not all of the elements that "should" be added, postponing others; as long as one continues (transfinitely) long enough, the process will stabilize, and the stable value will be the correct Γ^∞. I describe this situation by saying that, for monotone operators, the least fixed-point can be reached by an asynchronous iteration. One has freedom as to when to add the available new elements.

In the inflationary case, such an asynchronous iteration still leads to a fixed-point, but it need not be the correct one, the inflationary fixed-point Γ^∞ produced by the definition above. Inflationary fixed points must be computed by a synchronous iteration.

Despite this difference between the monotone and inflationary situations, it turns out that the two sorts of fixed-point constructions, when added to first-order logic, yield the same expressive power. This was proved for finite structures in [28] and in general in [33].

Least fixpoint logic is defined as the result of enhancing first-order logic with syntax for the least fixedpoints of positive formulas. That is, one adds a formation rule[5] saying that, if $\gamma(P, x)$ and $\phi(P)$ are formulas (in vocabulary $L \cup \{P\}$ where P is a k-ary predicate not in L, with k being the number of components of the sequence x) and if P occurs only positively in $\gamma(P, x)$, then

$$\text{Let } P(x) \leftarrow \gamma(P,x) \text{ then } \phi(P)$$

is a formula. Its semantics is defined as follows. Given a first-order L-structure A and values in A for all the free variables (namely the free variables in ϕ and the free variables other than x in γ), use the interpretation of γ to define, as above, a monotone operator Γ on k-ary relations on A, and use the least fixed-point Γ^∞ of that operator as the interpretation of P in $\phi(P)$. The requirement that P occur only positively in γ is imposed to ensure that Γ is monotone. The reason for not imposing monotonicity directly is that, unlike positivity, it is not a decidable property of formulas, and one wants the syntax to be decidable.

For inflationary fixpoint logic, one similarly extends the syntax except that positivity is no longer required. Instead, one uses the inflationary operator Γ defined by the formula $\gamma(P, \vec{x}) \vee P(\vec{x})$ to produce the intended interpretation Γ^∞ for P in $\phi(P)$.

A famous result of Immerman [31] and Vardi [39] says that first-order logic extended by either of these fixed-point constructs captures polynomial time on (linearly) ordered structures. "Capturing" here means that

- for every L-sentence ϕ of the logic, the class of its ordered finite models is recognizable in polynomial time, by a program that can be computed from ϕ, and
- every polynomial-time recognizable class of finite ordered L-structures that is closed under isomorphism is the class of models of some L-sentence of the logic.

The restriction to ordered structures here is essential. It means that the language L must have a binary relation symbol that is interpreted, in all the structures under consideration, as a linear ordering of the whole underlying set of the structure. Without this restriction, the Immerman–Vardi theorem fails badly. For example, on sets with no additional relations and functions (so only equality is available for building atomic formulas), first-order logic with (either) fixed-point operator cannot express "the cardinality of the set is even."

This deficiency in the case of unordered structures was one of the motivations for the study of a further extension of the logic, allowing some form of counting. The resulting logics can, by design, express properties like "the cardinality of the set is even," but some more subtle polynomial-time computable properties remain inexpressible. For example, Cai, Fürer, and Immerman [14] produced two infinite sequences of graphs, G_n and H_n of size linear in n, that are not isomorphic but are so similar that every sentence of fixed-point logic plus counting fails, once n is large enough, to distinguish G_n from H_n.

A key to establishing such results is the fact that fixpoint logic can be embedded in the infinitary logic $L_{\infty,\omega}$ (which allows conjunctions and disjunctions of any set of formulas, not just finite sets) with the restriction that one allows only those $L_{\infty,\omega}$-formulas in which the total number of variables is finite. The total number here refers to both free and bound variables, but it is permitted to re-use the same variable in different (even nested) subformulas. For more information about these matters, see Otto's book [34]. It should be noted that infinitary logics, in some cases allowing not only infinite conjunctions and disjunctions but also infinite blocks of quantifiers, had been studied much earlier for purely mathematical purposes, with no expectation of applicability to computing or to finite models; see for example [32].

5. Gurevich conjecture and choiceless polynomial time

Gurevich [26] conjectured that the restriction to ordered structures in the Immerman–Vardi theorem is absolutely essential, meaning that *no* logic can capture polynomial-time computability on all (not necessarily ordered) finite structures. To make the conjecture precise, it is necessary to say what counts as a logic. Gurevich provided a precise definition, broad enough to allow anything that might reasonably be proposed as a logic for this purpose. For this conjecture, a logic must provide, for each finite first-order vocabulary (also called language or signature) L, a recursive set of L-sentences and a recursive satisfaction relation between finite first-order L-structures and L-sentences, invariant under isomorphism of structures. The notion of capturing polynomial time is defined for such logics just as in the case of fixed-point logics above.

Notice that this definition of logic allows not only the things one usually calls logics but also "logics" where the "sentences" are actually computer programs and "satisfaction" is defined in terms of running the program on an input structure. The definition is so broad that one might think at first that one could simply take "sentence" to mean a polynomial-time program (perhaps with a built-in clock to ensure that it takes only polynomial time) and thereby produce a logic capturing polynomial time. Fortunately for the conjecture, the definition is not quite that broad. One must limit the sentences to ones that are invariant under isomorphism of structures, and the set of all such invariant programs is not recursive.

Gurevich's conjecture is still an open problem. A plausible challenge to the conjecture is given by the logic of choiceless polynomial time (CPT)˜ introduced in [10] plus counting. The ideas behind CPT˜ (without counting) are:

- The input to a computation is a first-order structure.
- Allow "arbitrary" data structures.
- Allow parallelism.
- Prohibit arbitrary choices (or ordering unless provided by the structure itself).
- Polynomially bound the total work (honestly measured).

These ideas are formalized in [10] in terms of abstract state machines working, in polynomial time, over HF(A), the universe of hereditarily finite sets over the input structure A. The use of HF(A) is a way of making arbitrary data structures available, since such structures can be coded set-theoretically.

By itself, CPT˜ is quite weak. It cannot count. In fact it satisfies a zero-one law [35] (or see [8]). So naturally one adds counting to the logic. Gurevich's conjecture predicts that there is a polynomial-time computable, isomorphism-

invariant property of first-order structures (for some finite language) that cannot be expressed in CPT˜ with counting. No example of such a property is known yet. There have been two serious attempts, with interesting outcomes.

First, Gurevich and I expected that the bipartite matching problem, given a bipartite graph does it admit a perfect matching (a set of edges such that every vertex is incident to exactly one of them), would be inexpressible in CPT˜ with counting. When we mentioned this to Shelah, he came up (the next day) with a very clever definition of it in CPT˜ plus counting [11]. Very recently, Anderson, Dawar, and Holm [3] have developed powerful techniques that yield the same result for arbitrary, not necessarily bipartite graphs.

Second, there is the polynomial-time solvable problem of recognizing, given two of the Cai–Fürer– Immerman graphs (from [14]) mentioned above, whether they are isomorphic. This was not known to be expressible in CPT˜ plus counting until Rossman exhibited a very clever definition of it in CPT˜ (without counting!). A companion result of Dawar and Richerby says that no such definition, even with counting, is possible if one uses hereditarily finite sets of bounded rank. See [23] for these results.

In both of these cases, we did not get an example of the sort predicted by Gurevich's conjecture; the problems turned out to be expressible in CPT˜ with counting. Nevertheless, in my opinion, the need for (very different) clever ideas in the proofs supports the conjecture.

6. Existential fixed-point logic

In addition to adding new constructs, like fixed-point operators, counting, and infinitary connectives, to first-order logic, one can also consider removing constructs, or combining some additions and some removals. In my opinion, a particularly interesting and fruitful logic of this last sort is existential fixed-point logic (∃FPL). It has arisen in the work of Aczel [2] on abstract computability, in the work of Chandra and Harel [15] on database queries, and in work of Gurevich and me [7] motivated by the study of Hoare logic.

In rough terms, ∃FPL is obtained from first-order logic by adjoining the least fixed-point operator but removing the universal quantifier. For this to make sense, one must ensure that universal quantification is not reintroduced by combining existential quantification and negation. The formal definition begins with vocabularies in which the relation symbols are classified as *negatable* or *positive*. Then one forms

- terms and atomic formulas as usual in first-order logic,
- negations only of atomic formulas that begin with a negatable relation symbol,
- ∧, ∨, ∃ as usual, and
- least fixed points, as in Section 4, including simultaneous fixed points as in the footnote there, requiring that the predicate symbols used for the iteration be positive.

This logic has numerous pleasant properties, including the following.

(1) It captures polynomial time on structures with the successor relation.
(2) It is appropriate for the Hoare logic of asserted programs.
(3) Satisfaction of a sentence in a structure depends on only a finite part of the structure.
(4) The iterations leading to the least fixed-points used in the semantics stabilize at the first transfinite level, ω, if not sooner.

A few clarifying remarks are in order here. In (1), the structures are linearly ordered, but the vocabulary is required to have a name for the immediate successor relation rather than the linear ordering as in the Immerman–Vardi theorem. The point here is that ∃FPL can define the ordering from the successor relation but not vice-versa. This is the exact opposite of what happens in first-order logic. In (2), "appropriate" means that, if one uses ∃FPL in place of first-order logic as the assertion language for Hoare logic, then one gets the analog of Cook's completeness theorem [20] without the need for an expressivity hypothesis. For more details about these, and some additional pleasant properties, see [6] and the references there.

The computational properties of ∃FPL are also interesting. It is shown in [7] that:

- validity of ∃FPL formulas is complete Σ_1^0,
- satisfiability of ∃FPL formulas is complete Σ_1^0,
- validity of sequents (expressions of the form $\forall \vec{x}\,(\varphi \quad \psi \rightarrow)$, where ϕ and ψ are ∃FPL-formulas but the whole sequent is not because of ∀ and →) is complete Π_2^0, and
- the consequence relation among sequents is complete Π_1^1.

The first two of these may look strange, as validity and satisfiability ordinarily have dual complexities. The reason for that, however, involves the possibility of negating arbitrary formulas, and that possibility is absent in ∃FPL.

The distinction between negatable and positive relation symbols has a natural interpretation in terms of databases. To explain this it is useful to consider another pleasant property of ∃FPL-sentences, namely that their truth is preserved by homomorphisms of structures; if $h : A \rightarrow B$ is a homomorphism and an ∃FPL-sentence ϕ is true in A, then it is also true in B. The appropriate definition of "homomorphism" here is that h must commute with the interpretations of function symbols, must preserve in the forward direction the interpretations of positive relation symbols, and must preserve in both directions the interpretations of negatable relation symbols. Such a homomorphism can be viewed intuitively as the evolution of a database, from A to B, when more information becomes available without any change in the real world. One might learn of new elements (when h is not surjective), one might learn that some elements are equal (when h is not injective),[6] and one might learn that a positive predicate holds of some additional elements. But one cannot learn that a negatable predicate holds of a previously known element; we cannot have $P^B(h(a))$ without $P^A(a)$ when P is negatable. In this sense, negatability of a predicate is a sort of closed-world assumption; if we know of an element but we don't know that it satisfies P, then we actually know that it doesn't satisfy P.

It is natural, in this connection, to consider another sort of closed-world assumption, namely that we already know about all elements of some sort, i.e., that the development from A to B will not introduce any new elements of that sort. So homomorphisms should be surjective on that sort. On the syntactic side, this corresponds to allowing universal quantification for that sort. For details about this and about whether pleasant properties of ∃FPL can survive this modification, see [6].

7. Fixed-point deduction

Fixed-point constructions first attracted my interest far from computer science. The motivation came from set theory, specifically from a comment in [36, Section 9.10] about large cardinals as a way to strengthen the axioms of set theory. Shoenfield points out there that:

> [T]he existence axioms of ZFC are very deficient in some respects. For example, the subset axioms do not really guarantee the existence of all subsets of x, but only of those subsets which can be described in ZFC (using parameters). If we introduced symbols for new operations which cannot be defined in ZFC, we would increase our ability to describe sets and hence increase the power of the subset (and replacement) axioms. This appears to be a natural approach; but so far no one has been able to propose any suitable operations.

My reaction to this was that there is a very natural operation to add, namely a definition of truth, or more precisely a satisfaction predicate for the language of ZFC. One could add a new predicate symbol for the satisfaction predicate, add axioms saying that it obeys the usual inductive clauses defining satisfaction (for the original language of ZFC, not including the new predicate), and extend the subset and replacement axioms to formulas that contain the new symbol. That is a genuine strengthening of ZFC, because it proves the consistency of ZFC, but it seems just as believable as ZFC itself.

Once one has the idea of adding this particular inductively defined predicate, as a new primitive notion, and adding its inductive "definition" as new axioms, one can hardly avoid asking, why just this particular induction? Why not arbitrary least fixed-point definitions? Why not include this capability as part of the underlying logic, so that, as soon as one has defined the necessary ingredients for individual steps of an induction, the final result, the fixed point, becomes available? That amounts to using first-order logic augmented with the least fixed-point operator.

There is, of course, a good reason for not doing this. A great advantage of first-order logic for foundational purposes is the existence of a sound and complete deductive system. One can confidently work with formal proofs. Nothing like that is possible for a system with a least-fixed-point operator. The definability of transitive closures in such a system guarantees the failure of the compactness theorem, while the existence of a sound, complete, and finitary deductive system implies compactness.

One can, of course, give up finitariness; allow the deductive system to have rules with infinitely many premises. This is the approach taken in [18] for ∃FPL. I want to raise another possibility, namely to give up completeness. The idea will be to simply write down finitary rules and axioms describing what a least fixed-point should look like, and see what can be deduced. My proposal is to add the following to a standard formalization of first-order logic.

- For any formula $\gamma(P, x_1, \ldots, x_k)$ in which the k-ary predicate variable P occurs only positively, introduce a new k-ary predicate symbol C (intended to denote the least fixed-point of the operator Γ defined by γ). • Add the axiom

$$\gamma(C,x) \rightarrow C(x)$$

  saying that C is closed under Γ.
- Add the axiom schema

$$\forall \vec{x}\,\big(\gamma(\psi(-),\vec{x}) \rightarrow \psi(\vec{x})\big) \rightarrow \forall \vec{x}\big(C(\vec{x}) \rightarrow \psi(\vec{x})$$

  saying that C is least among (definable) predicates closed under Γ.

The ψ in the axiom schema is allowed to be any formula in the system, including formulas that involve C or involve further fixed-point predicates introduced by means of other γ's that might involve C. In other words, I want the schema to assert the leastness of the fixed-point C in the strongest available way.

Having given up completeness in favor of finitariness, we must ask how much has been lost. Some semantically valid facts in this system will be unprovable, but are there any interesting or important facts of this nature? Here is

a test question that I don't know the answer to, a quite natural (in my opinion) fact that I don't know how to deduce in this formal system — nor do I have a proof that it can't be deduced.

Suppose Γ is a monotone operator on binary relations such that, whenever P is the graph of a partial function then so is $\Gamma(P)$. It follows immediately, by induction on the ordinal α, that each stage Γ^α of the iteration is the graph of a partial function. In particular, so is the least fixed-point Γ^∞. Can this conclusion be derived in the formal system described above? That is, suppose γ is such that we have

$$\forall x \exists^{\leq 1} y\, P(x,y) \rightarrow \forall x \exists^{\leq 1} y\, \gamma(P,x,y).$$

Can one deduce, for the corresponding fixed-point predicate C, that

$$\forall x \exists^{\leq 1} y\, C(x,y)\ ?$$

Acknowledgements

This paper is an expanded version of the talk I gave at Logic Colloquium 2012 in Manchester. I thank the Programme Committee for inviting me to give that talk, and I especially thank Paola D'Aquino for encouraging me to write it up as a paper. I also thank G. Peter Scott for providing the information in Section 2 about the homeomorphism problem for 3-manifolds.

References

[1] Peter Aczel, An introduction to inductive definitions, Chapter C.7, in: K.J. Barwise (Ed.), Handbook of Mathematical Logic, in: Studies in Logic and the Foundations of Mathematics, vol. 90, North-Holland, 1977, pp. 739–782.
[2] Matthew Anderson, Anuj Dawar, Bjarki Holm, Maximum matching and linear programming in fixed-point logic with counting, in: Proc. 28th IEEE Symposium on Logic in Computer Science, LICS, 2013, pp. 173–182.
[3] Laurent Bessières, Gérard Besson, Sylvain Maillot, Michel Boileau, Joan Porti, Geometrisation of 3-Manifolds, EMS Tracts in Mathematics, vol. 13, European Mathematical Society, 2010.
[4] Andreas Blass, Abstract state machines and pure mathematics, in: Y. Gurevich, P. Kutter, M. Odersky, L. Thiele (Eds.), Proceedings of the International Workshop on Abstract State Machines: Theory and Applications, ASM 2000, Monte Verità, Switzerland, March 2000, in: Lecture Notes in Computer Science, vol. 1912, Springer-Verlag, 2000, pp. 9–21.
[5] Andreas Blass, Existential fixed-point logic, universal quantifiers, and topoi, in: A. Blass, N. Dershowitz, W. Reisig (Eds.), Fields of Logic and Computation. Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday, in: Lecture Notes in Computer Science, vol. 6300, Springer-Verlag, 2010, pp. 108–134.
[6] Andreas Blass, Yuri Gurevich, Existential fixed-point logic, in: E. Börger (Ed.), Computation Theory and Logic, in: Lecture Notes in Computer Science, vol. 270, Springer-Verlag, 1987, pp. 20–36.
[7] Andreas Blass, Yuri Gurevich, Strong extension axioms and Shelah's zero-one law for choiceless polynomial time, J. Symbolic Logic 68 (2003) 65–131.
[8] Andreas Blass, Yuri Gurevich, Abstract state machines capture parallel algorithms, ACM Trans. Comput. Log. 4 (2003) 578–651. Correction and extension, ACM Trans. Comput. Log. 9 (2008), article 19.
[9] Andreas Blass, Yuri Gurevich, Saharon Shelah, Choiceless polynomial time, Ann. Pure Appl. Logic 100 (1999) 141–187.
[10] Andreas Blass, Yuri Gurevich, Saharon Shelah, On polynomial time computation over unordered structures, J. Symbolic Logic 67 (2002) 1093–1125.
[11] W.W. Boone, F.B. Cannonito, R.C. Lyndon (Eds.), Word Problems. Decision Problems and the Burnside Problem in Group Theory, Studies in Logic and the Foundations of Mathematics, vol. 71, North-Holland, 1973.
[12] Egon Börger, Erich Grädel, Yuri Gurevich, The Classical Decision Problem, Perspectives in Mathematical Logic, SpringerVerlag, 1997.
[13] Jin-Yi Cai, Martin Fürer, Neil Immerman, An optimal lower bound on the number of variables for graph identification, Combinatorica 12 (1992) 389–410.
[14] Ashok Chandra, David Harel, Horn clause queries and generalizations, J. Log. Program. 2 (1985) 1–15.

[15] Alonzo Church, A note on the Entscheidungsproblem, J. Symbolic Logic 1 (1936) 40–41. Correction in J. Symbolic Logic 1 (1936) 101–102.
[16] Edgar F. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (6) (1970) 377–387.
[17] Kevin J. Compton, A deductive system for existential fixed-point logic, J. Logic Comput. 2 (1993) 197–213.
[18] Stephen A. Cook, The complexity of theorem-proving procedures, in: Proc. Third Annual ACM Symp. on the Theory of Computing, STOC, ACM, 1971, pp. 151–158.
[19] Stephen A. Cook, Soundness and completeness of an axiom system for program verification, SIAM J. Comput. 7 (1978) 70–90.
[20] Martin Davis, Yuri Matijasevič, Julia Robinson, Hilbert's tenth problem. Diophantine equations: positive aspects of a negative solution, in: F.E. Browder (Ed.), Mathematical Developments Arising from Hilbert Problems, in: Proc. Sympos. Pure Math., vol. XXVIII, Amer. Math. Soc., 1976, pp. 323–378.
[21] Anuj Dawar, Yuri Gurevich, Fixed point logics, Bull. Symbolic Logic 8 (2002) 65–88.
[22] Anuj Dawar, David Richerby, Benjamin Rossman, Choiceless polynomial time, counting, and the Cai–Fürer–Immerman graphs, Ann. Pure Appl. Logic 152 (2008) 31–50.
[23] Max Dehn, Über unendliche diskontinuierliche Gruppen, Math. Ann. 71 (1911) 116–144.
[24] Michael R. Garey, David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., 1979.
[25] Yuri Gurevich, Logic and the challenge of computer science, in: E. Börger (Ed.), Current Trends in Computer Science, Computer Science Press, 1988, pp. 1–57.
[26] Yuri Gurevich, Sequential abstract state machines capture sequential algorithms, ACM Trans. Comput. Log. 1 (2000) 77–111.
[27] Yuri Gurevich, Saharon Shelah, Fixed-point extensions of first-order logic, Ann. Pure Appl. Logic 32 (1986) 265–280.
[28] David Hilbert, Mathematische Probleme, Nachr. Akad. Wiss. Göttingen Math.-Phys. Kl. II (1900) 253–297, English translation in Bull. Amer. Math. Soc. 8 (1902) 437–479 and in: F.E. Browder (Ed.), Mathematical Developments Arising from Hilbert Problems, in: Proc. Sympos. Pure Math., vol. XXVIII, Amer. Math. Soc., 1976, pp. 1–34.
[29] David Hilbert, Wilhelm Ackermann, Grundzüge der Theoretischen Logik, Springer-Verlag, 1928.
[30] Neil Immerman, Relational queries computable in polynomial time, Inf. Control 68 (1986) 86–104. Preliminary version in: Proc. 14th ACM Symp. on Theory of Computing, STOC, 1982, pp. 147–152.
[31] Carol R. Karp, Languages with Expressions of Infinite Length, North-Holland, 1964.
[32] Stephan Kreutzer, Expressive equivalence of least and inflationary fixed-point logic, Ann. Pure Appl. Logic 130 (2004) 61–78, LICS 2002 selected paper issue.
[33] Martin Otto, Bounded Variable Logics and Counting. A Study in Finite Models, Lecture Notes in Logic, vol. 9, SpringerVerlag, 1997.
[34] Saharon Shelah, Choiceless polynomial time logic: inability to express, in: P. Clote, H. Schwichtenberg (Eds.), Computer Science Logic 2000, in: Lecture Notes in Computer Science, vol. 1862, Springer-Verlag, 2000, pp. 72–125.
[35] Joseph R. Shoenfield, Mathematical Logic, Addison–Wesley, 1967.
[36] Jürgen Teich, Philipp Kutter, Ralph Weper, Description and simulation of microprocessor instruction sets using ASMs, in: Y. Gurevich, P. Kutter, M. Odersky, L. Thiele (Eds.), Proceedings of the International Workshop on Abstract State Machines: Theory and Applications, ASM 2000, Monte Verità, Switzerland, March 2000, in: Lecture Notes in Computer Science, vol. 1912, Springer-Verlag, 2000, pp. 266–286.
[37] Alan Turing, On computable numbers, with an application to the 'Entscheidungsproblem', Proc. Lond. Math. Soc. (2) 42 (1937) 230–265. Correction in Proc. Lond. Math. Soc. (2) 43 (1937) 544–546.
[38] Moshe Vardi, The complexity of relational query languages, in: Proc. 14th ACM Symp. on Theory of Computing, STOC, 1982, pp. 137–146.