

# A NEW KIND OF SOLUTION, AND AN APPLICATION TO RECURRENCE RELATIONS

HENRY YUEN

ABSTRACT. In this essay, I philosophize about the meaning of “solution” in mathematics, and apply this philosophy to generalize recurrence relations. A complexity theoretic result is proved about recurrence relations.

## 1. THE EVOLUTION OF SOLUTION

Throughout the history of mathematics, a central goal of its practitioners has been to find simple expressions for solutions to equations. Although there is no rigorous definition of “simple expression”, mathematicians have nonetheless spent enormous amounts of effort seeking solutions to their analytical problems that satisfy their aesthetic of simplicity. This aesthetic has evolved over time. In antiquity, “simple” meant *rational*: the ancient Greeks conducted a (misguided) search for two integers  $a$  and  $b$  such  $(a/b)^2 = 2$ . Later, “simple” meant having an *algebraic expression*, which involve only addition, multiplication, and  $n$ th-roots. For centuries, mathematicians sought the elusive quintic formula, an algebraic expression for the roots of a degree 5 polynomial. One of the greatest triumphs of mathematics was Galois’s discovery that there is no such expression, that in general polynomials are too *complex* to be subject to easy solution.

More recently, “simple” has been equated with *closed form* expressions, which has a rather nebulous definition. Generally, closed form expressions are composed of so-called “elementary” operations: the usual arithmetic operations, radicals, exponentiation and logarithms (and thus trigonometric functions are included). The three-body problem is an iconic example of a system of equations resisting all attempts to find a closed form solution. The problem statement is simple: given the initial state (positions and velocities) of 3 point masses experiencing mutual gravitational attraction, determine their state at a later time. The three-body problem has notoriously consumed mathematicians of the likes of Newton, Lagrange, Euler, Hilbert, Poincaré. Their efforts have led to a bountiful harvest in terms of a deeper understanding of analysis and dynamical systems, but the holy grail - the closed form solution - to the three-body problem remains out of reach. Indeed, a closed form solution is probably too much to hope for.

As our mathematical understanding advances, so does the reach of “simple”. Each form of simplicity, whether it meant rational, or algebraic, or closed-form, is a reflection of what mathematicians considered amenable to computation at the time. The Greeks abhorred the irrationality of  $\sqrt{2}$ , because it meant that geometrical computations would fail to be *exact*. Before calculus, non-algebraic expressions for roots of polynomials was anathema to mathematicians. Without the use of high speed computers, calculating the precise trajectory of three heavenly objects seemed intractable.

Today, none of these problems give mathematicians headaches anymore, mostly thanks to Moore's Law. Computers barely work up a sweat when dealing with the three-body problem; astrophysicists now routinely simulate the dynamics of entire *galactic clusters*. Nearly every area of scientific and industrial computing performs massive numerical linear algebra, which can involve finding the roots of polynomials of degree  $1 \times 10^6$ . Today, "simple" means something like "solvable by computers within a reasonable amount of time".

Fortunately, this has a formal definition in mathematics: it is also known as *polynomial-time computability*. The formalization of the notion of efficient computation is at heart of computational complexity, the branch of theoretical computer science that studies the *difficulty* of solving mathematical problems. Just as zoologist attempts to provide a complete classification of life on Earth, so does a computational complexity theorist try to categorize mathematical problems by the computational resources needed to solve them. Perhaps the most important category is the set of problems that are solvable by computers in polynomial time, or  $P$ . That is, the number of steps that are required to solve a problem in  $P$  scales as a polynomial in the length of the problem instance you're trying to solve.

Though they were not aware of it, the Greeks, the predecessors of Galois, mathematicians trying to compute the dynamics of 3 bodies - they were seeking limited forms of polynomial time algorithms for their respective problems. One could say that, before electronic computers, the working notion of polynomial time computation *was* that involving only plus, times,  $n$ th-roots, and perhaps exponentiation and logarithms. Now that we do have computers at our disposal, "simple" now encompasses the much more general notion of  $P$ . And because of the Church-Turing Thesis, I argue that  $P$  is probably the most general notion of "simple"<sup>1</sup>. The buck stops with  $P$ : if your laptop can't solve it efficiently, neither can any other mathematician<sup>2</sup>.

Many scientists and engineers continue to seek solutions to their problems that fit the traditional notion of simple; closed form expressions are still regarded as the gold standard for tractable solutions. Other than its historical importance, though, the criteria for closed-formness is largely arbitrary. What is so special about computation methods that only involve  $+$ ,  $\times$ , and  $\sqrt{\quad}$ ? Why should computing  $\int e^{x^2} dx$  be any more difficult than computing  $\int e^x dx$ ? Why can't a polynomial time algorithm count as a solution to a problem?

The age of the closed form expression is over. Modern problems are usually too complex to admit nice, clean formulas. Instead, they require a more general algorithmic approach, and now issues of computational complexity and numerical accuracy take center stage. For solvers of mathematical problems, it is important to understand what can be computed efficiently and accurately. And just as the the discovery of the irrationality of  $\sqrt{2}$ , Galois theory, and the 3-body problem have led to major leaps in mathematics, so will the study of algorithms and complexity.

## 2. AN APPLICATION: THE RECURRENCE RELATION

Let's apply this philosophy to recurrence relations. Recurrence relations are equations that define real number sequences recursively, usually with some specified initial condition.

---

<sup>1</sup>Let's leave quantum computers out of the picture for a moment.

<sup>2</sup>A nod to Kamal Jain's pithy quote.

One simple recurrence relation is

$$a_{n+1} = 2a_n, a_0 = 1$$

which gives rise to the sequence 1, 2, 4, 8,  $\dots$ . An important example is the logistic map:

$$x_{n+1} = rx_n(1 - x_n).$$

Depending on your choice of  $r$  and  $x_0$ , the generated sequence can exhibit chaotic behavior.

While the first recurrence relation given has a traditional closed form solution ( $a_n = 2^n$ ), the second one is not known to have one. Indeed, it is expected not to. In light of the preceding philosophy, this leads to the following question: even if the logistic map doesn't admit a traditional closed form solution, can  $x_n$  still be efficiently computed? I don't know if anyone knows the answer to this interesting question, but the following statement is true:

**Theorem 1** (Informal). *Polynomial-time definable recurrence relations are not generally polynomial-time solvable, even if the solution is succinct, unless  $P = PSPACE$ .*

We will define what we mean by a “polynomial-time definable recurrence relation”, or by a “succinct solution”, soon. But, in the same spirit of proving that  $\sqrt{2}$  is irrational, or that the quintic polynomial is not generally solvable by radicals, this theorem is an impossibility result (modulo a widely-believed assumption of computational complexity). It is perhaps the most general possible, too, because a polynomial-time definable recurrence relation with a succinct solution can be solved in  $PSPACE$ . Let's give proper definitions now. Some conventions:  $\mathbb{N}$  includes 0; when we write  $|x|$  for some  $x \in \mathbb{N}$ , we mean the number of bits required to represent  $x$ .

**Definition 1.** A polynomial-time definable recurrence relation (PTDRR) is a pair  $(G, s)$  where  $G : \mathbb{N} \rightarrow \mathbb{N}$  is a polynomial-time computable function, and  $s \in \mathbb{N}$ . A *solution* to a PTDRR is a function  $f : \mathbb{N} \rightarrow \mathbb{N}$  such that  $f(0) = s$ ,  $f(n+1) = G(f(n))$ , for all  $n \in \mathbb{N}^3$ .

**Theorem 2.** *For all PTDRR  $(G, s)$ , there exists a unique solution, and it is computable in  $EXP$ .*

*Proof.* A solution  $f$  satisfies  $f(n) = \overbrace{G(G(\dots G(|s|)\dots))}^{n \text{ times}}$ , and since the initial condition  $f(0) = s$  is given,  $f(n)$  is determined for all  $n$ , and hence unique. Furthermore, for  $j = 1, 2, \dots, n$ ,  $|f(j)| \leq \overbrace{q(q(\dots q(|s|)\dots))}^{n \text{ times}} \leq 2^{O(n)}$ , where  $q$  is a polynomial such that  $G(y)$  halts in  $q(|y|)$  time. Thus, iterated application of  $G$  takes  $2^{O(n)}$  time.  $\square$

Notice that not all PTDRRs admit polynomial-time computable solutions. Consider  $(G, s)$ , where  $G(y) = y^2$ , and  $s = 2$ . The solution is  $f(n) = 2^{2^n}$ , and it takes exponential time to even write out the answer for each  $n$ ! What if we limited our solutions to have polynomial-sized descriptions? Perhaps then we would have a chance at polynomial-time solvability. It turns out that this is too much to hope for.

**Definition 2.** A solution  $f(n)$  of a PTDRR  $(G, s)$  is *succinct* if there exists a polynomial  $p$  such that  $|f(n)| \leq p(|n|)$ .

<sup>3</sup>For simplicity, I have defined  $f(n+1)$  to depend only on  $f(n)$ , but this could be easily generalized so that  $f(n+1)$  depends on more terms from  $\{f(0), f(1), \dots, f(n)\}$  (or perhaps some polynomial-sized subset).

**Theorem 3.** A PTDRR  $(G, s)$  with a succinct solution is computable in PSPACE.

*Proof.* Since  $G$  runs in polynomial time, and the length of its input is at most some polynomial in  $|n|$ , each run of  $G$  takes polynomial space. Thus, the iterated composition of  $G$  requires only polynomial space.  $\square$

**Definition 3.** A PTDRR  $(G, s)$  is *polynomial-time solvable* if the solution  $f(n)$  can be computed in polynomial time in  $|n|$ .

**Theorem 4.** There exists a PTDRR  $(G, s)$  with a succinct solution that is not computable in polynomial time, unless  $P = PSPACE$ .

The proof idea is to exhibit a polynomial-time definable recurrence relation  $(G, s)$  such that the solution  $f$  encodes the computational history of a Turing Machine that solves TQBF (“True Quantified Boolean Formula”, a PSPACE-complete problem) on *every* string.  $G$  will be a very generic function that essentially takes an instantaneous description of a Turing Machine and advances the state of the TM one step. If  $f$  is polynomial-time computable, then so is  $P = PSPACE$ , which is considered highly unlikely by complexity theorists.

*Proof.* Let  $M$  be a polynomial-space Turing Machine that, on input  $x$ , accepts if and only if  $x$  represents, in some efficient encoding, a quantified boolean formula (a boolean formula where every variable is quantified using either universal or existential quantifiers) that is satisfiable. Let  $q$  be a polynomial such that  $M$  runs in  $2^{q(|x|)}$  time.

For all  $x \in \mathbb{N}$ ,  $1 \leq t \leq 2^{q(|x|)}$ , let  $D_M(x, t)$  be the instantaneous description of  $M$  on input  $x$  at time  $t$ . We will associate each input  $x$  and instantaneous description  $D_M(x, t)$  with a natural number in the following manner:

- (1) For  $1 \leq t \leq 2^{q(|0|)}$ , associate  $D_M(0, t)$  with natural number  $t - 1$ .
- (2) For  $1 \leq t \leq 2^{q(|1|)}$ , associate  $D_M(1, t)$  with natural number  $2^{q(|0|)} + t - 1$ .
- (3) In general, for arbitrary input  $x \geq 2$ ,  $1 \leq t \leq 2^{q(|x|)}$ , associate  $D_M(x, t)$  with

$$t - 1 + (x - 2^{|x|-1})2^{q(|x|)} + \sum_{j=1}^{|x|-1} 2^{q(j)}$$

Call this mapping  $\lambda(x, t)$ . Note that  $\lambda(x, t)$  can be computed in time polynomial in  $|x|$  and  $|t|$ , and for all  $x$ ,  $1 \leq t \leq 2^{q(|x|)}$ ,  $\lambda$  is one-to-one and onto. Thus for all  $n$ ,  $\lambda^{-1}(n)$  is well defined.

Consider the sequence  $f(n) = \langle x, t, D_M(x, t) \rangle$ , where  $(x, t) = \lambda^{-1}(n)$  and  $\langle \cdot \rangle$  is some efficient encoding scheme of strings into integers.  $f$  is a solution to the polynomial-time definable recurrence relation  $(G, s)$ , where  $s = \langle 0, 1, D_M(0, 1) \rangle$ , and  $G$  has the following behavior:

$$G(\langle x, t, D_M(x, t) \rangle) = \begin{cases} \langle x, t + 1, D_M(x, t + 1) \rangle & \text{if } t < 2^{q(|x|)} \\ \langle x + 1, 1, D_M(x + 1, 1) \rangle & \text{if } t = 2^{q(|x|)} \end{cases}$$

$f$  also has polynomial length.

Now suppose  $f$  was computable in polynomial time. We show that this would imply  $P = PSPACE$ : Let  $L \in PSPACE$ . Because of PSPACE-completeness of TBQF, there exists a polynomial time computable function  $R$  such that for all  $y \in \{0, 1\}^*$ ,  $y \in L$  if and only

if  $R(\mathbf{y}) \in \text{TBQF}$ . To decide if a string  $\mathbf{y} \in L$ , compute  $f(\lambda(R(\mathbf{y}), 2^{q(|\mathbf{y}|)}))$ . The output will be  $\langle R(\mathbf{y}), 2^{q(|\mathbf{y}|)}, D_M(R(\mathbf{y}), 2^{q(|\mathbf{y}|)}) \rangle$ , and we can determine if  $D_M(R(\mathbf{y}), 2^{q(|\mathbf{y}|)})$  describes an accepting state of  $M$ . If so, then  $\mathbf{y} \in L$ . This procedure to decide  $L$ , takes time polynomial in the length of  $\mathbf{y}$ .

□

### 3. INTERPRETATION CONSIDERATIONS

We considered a new notion of recurrence relation, one that inherently computational in nature, and saw an intractability result. I will try to argue that this generalization is an appropriate one, and provides a perspective from which interesting research could be pursued.

There are two parts to this new notion of recurrence relation: polynomial-time definability and polynomial-size solvability (succinctness). Both parts are intended to capture *tractable* recurrence relations. I don't mean recurrence relations with tractable solutions (we just saw a PTDRR with a succinct solution that still isn't efficiently solvable). I mean, if we want to even *begin* to consider solving a recurrence relation, we should at least insist that it be feasible to brute force! What hope could there be in computing values of  $f(\mathbf{n})$  when the recurrence relation  $f(\mathbf{n} + 1) = G(f(\mathbf{n}))$  involves a  $G$  that requires exponential time to compute? All natural examples of recurrence relations that I can think of are all polynomial-time definable. If pressed, I could lamely present you with the following *exponential*-time definable recurrence relation  $(G, s)$ , where  $G(\mathbf{y}) = \langle \mathbf{y}, 1 \rangle$  if  $\mathbf{y}$  encodes some  $m \times m$  chess game, and white has a winning strategy. Otherwise,  $G(\mathbf{y}) = \langle \mathbf{y}, 0 \rangle$ . Computing winning strategies for  $m \times m$  chess is known to be EXPTIME-complete; therefore, even if you're given  $f(\mathbf{n})$ , trying to compute  $f(\mathbf{n} + 1)$  will surely take longer than the life of the universe even for moderate  $\mathbf{n}$ . A crass example, perhaps, but I am very interested to see if anyone can give an example of a "natural" recurrence relation that is not polynomial-time definable!

Succinctness is another important aspect of tractable recurrence relations (again, I don't mean efficiently solvable here). Suppose we were given an oracle to a solution for a recurrence relation. Even if we've saved on the time it took to compute  $f(\mathbf{n})$ , the oracle wouldn't be very useful if outputting  $f(80)$  required more bits of space than the number of elementary particles in the visible universe! In that case, the recurrence relation is probably uninteresting to most people.

Wait! What about  $(G, s)$  where  $G(\mathbf{y}) = \mathbf{y}^2$ , and  $s = 2$ ? This seems like a perfectly interesting recurrence relation. Recall that  $f(\mathbf{n}) = 2^{2^n}$ , which requires  $2^n$  bits to write.  $f(\mathbf{n})$  isn't succinct (and hence not polynomial-time computable), yet it intuitively seems "simple". After all, we just wrote it down! While the given  $(G, s)$  might not be succinct, there is a closely related PTDRR  $(H, t)$  that *is* succinct, and even polynomial-time solvable! Instead of numerically squaring the input,  $H$  will square the input *symbolically*. If we set  $t = "2"$  (i.e., the number 2 encoded in ASCII), and if  $H$  acted like Mathematica<sup>4</sup>, then a solution to the PTDRR would be  $f(\mathbf{n}) = "2^{2^n}"$ . This sort of symbolic manipulation is how we perform mathematics anyways. When computing the 20th element of the sequence defined by  $f(\mathbf{n} + 1) = f(\mathbf{n})^2, f(0) = 2$ , we don't whip out the calculator and start repeated squaring - we simply just write down  $2^{2^{20}}$ . Despite the mega astronomical sizes of the output

<sup>4</sup>Indeed, running `RSolve[{a[n + 1] == a[n]^2, a[0] == 2}, a[n], n]` in Mathematica - or entering `a[n+1] = a[n]^2, a[0] = 2` in Wolfram Alpha - will solve the recurrence relation in polynomial time!

of  $f(\mathbf{n})$ , it is highly compressible. This leads to yet another question: is there a connection between compressibility of solutions to recurrence relations and the existence of traditional closed-form solutions? I do not know.

PTDRRs are extremely expressive, more than one would initially expect. For instance, one can write many systems of differential equations in the form of a PTDRR. As a simple illustration, let us convert the differential equation  $\mathbf{dy}/\mathbf{dx} = \mathbf{y}$  into a PTDRR. Let  $\mathbf{G}(\mathbf{y}) = (1 + \delta)\mathbf{y}$  (for some small  $\delta$ ), and  $s = 1$ . The solution to  $f(\mathbf{n} + 1) = \mathbf{G}(f(\mathbf{n}))$  is  $f(\mathbf{n}) = (1 + \delta)^{\mathbf{n}}$ , which, if  $\mathbf{n} \sim 1/\delta$ ,  $f(\mathbf{n})$  closely approximates  $e^x$  (where  $x = \delta\mathbf{n}$ ), and  $\mathbf{y} = e^x$  is indeed the solution to the original differential equation. What I have just described here is a primitive version of numerical integration, or quadrature. This is how differential equations are generally solved in practice, anyhow! For MORE complicated examples such as the famous heat equation  $\partial\mathbf{u}/\partial\mathbf{t} - \alpha\nabla^2\mathbf{u} = 0$ , advanced numerical integration methods like Gaussian quadrature could be used to improve accuracy.

Let me end with a few open questions:

**Which PTDRRs can be efficiently solved?** Though answering this in general is as hard as settling P vs NP, we can nonetheless try to answer this for special classes of PTDRRs. For example, we know from the theory of difference equations that homogenous linear recurrence relations of the form  $\mathbf{a}_n = \mathbf{c}_1\mathbf{a}_{n-1} + \mathbf{c}_2\mathbf{a}_{n-2} + \dots + \mathbf{c}_d\mathbf{a}_{n-d}$  can be solved in polynomial time (if  $d = O(\log n)$ ). Corless and Ilie [1] proved that a wide class of Differential Algebraic Equations can be solved in time polynomial in the number of bits of accuracy).

**Which PTDRRs can be efficiently “approximated”?** If certain PTDRRs don’t admit polynomial time solutions, maybe there is a notion of “approximate” solutions to recurrence relations. With the  $\mathbf{dy}/\mathbf{dx} = \mathbf{y}$  example above, the solution to the recurrence relation generally requires an exponential number of bits to describe. However, if we truncate  $f(\mathbf{n}) = (1 + \delta)^{\mathbf{n}}$  to only a polynomial number of bits, the rounded off  $\tilde{f}(\mathbf{n})$  would still approximately satisfy the recurrence relation  $\tilde{f}(\mathbf{n} + 1) \approx \mathbf{G}(\tilde{f}(\mathbf{n}))$ .

**What’s the relation between chaos and polynomial time solvability?** It is believed that the logistic map  $x_{n+1} = rx_n(1 - x_n)$  exhibits chaos for certain values of  $r$ . Does that imply it is not polynomial time solvable, or polynomial time approximable? Is there a general notion of a chaotic PTDRR (one that’s independent of a choice of metric, or encoding)? What is the analogue of sensitive dependence on initial conditions? A theory of PTDRRs could bridge the gap between the complex systems theory and computational complexity.

#### 4. ACKNOWLEDGMENTS

I thank Len Adleman and Anand Narayanan for their enlightening comments.

#### REFERENCES

- [1] Corless, R., Ilie, S. Polynomial Cost for Solving IVP for High-Index DAE *BIT Numerical Mathematics* 48:29-49, 2008.