LIMITATIONS ON OUR UNDERSTANDING OF THE BEHAVIOR OF SIMPLIFIED PHYSICAL SYSTEMS

by
Harvey M. Friedman
Distinguished University Professor of Mathematics, Philosophy, and Computer Science
Ohio State University
August 18, 2007

Abstract. Results going back to Turing and Gödel provide us with limitations on our ability to decide the truth or falsity of mathematical assertions in a number of important mathematical contexts.

There are two kinds of such limiting results that must be carefully distinguished. Results of the first kind state the nonexistence of any algorithm for determining whether any statement among a given set of statements is true or false.

Results of the second kind are much deeper and represent much greater challenges. They point to a specific statement A, among a given set of statements, where we can neither prove nor refute A using accepted principles of mathematical reasoning.

We give a brief survey of these limiting results. These include limiting results of the first kind: from number theory in mathematics, and from idealized computing devices in theoretical computer science.

The highlight of the talk is a discussion of limiting results of the first kind in the context of simplified physical systems; and a discussion of limiting results of the second kind. The simplified physical systems involve a small number of bodies, operating in potentially infinite one dimensional discrete space time.

AGENDA

1. EXAMPLES OF ALGORITHMS. Arithmetic ops, gcd, primality, factoring, solvability of equations.
2. NO ALGORITHMS. Robust model of computation. Behavior of abstract machines, solvability of equations.

1. EXAMPLES OF ALGORITHMS. Arithmetic ops, gcd, primality, factoring, solvability of equations.

I want to provide some significant context by discussing demonstrable successes before I discuss demonstrable limitations.

Long before there were any appropriate models of computation, there were actual interesting algorithms.

Schoolchildren are still taught the standard algorithms for adding, subtracting, and multiplying two integers given in base 10. They are also taught the standard division algorithm with remainder.

Algorithms for these standard arithmetic operations have been revisited in a very powerful way because of the computer revolution. There is a real need for optimizing speed as much as possible, for numerous applications.

New ideas were injected into this ancient business, taking advantage of the nature of actual circuit designs, including the capacity for parallelism.

Going beyond this universal staple, one of the most famous of all algorithms is the Euclidean algorithm from Greek times, which is believed to predate Euclid. The function of this algorithm is this. When given two two positive integers, n,m, in base 10, return the greatest common divisor of n,m in base 10.

The most obvious algorithm, when presented with n,m, maintains a number d which is the greatest integer found thus far to divide both n,m. We start with 1, which divides both n,m. Generally, we increment d by 1 and see if the result, d+1, divides both n,m. If so, we replace d by d+1, and pass on to d+1. If not, we leave d alone, and pass on to d+2.

We continue in this way. But how do we know when to stop? Conservatively, we can stop after we have dealt with max(n,m).

Of course, this is about the "worst possible algorithm" in terms of computing resources, imaginable. The Greeks did far better, in the following clever way.

If n = m, then gcd(n,m) = n. So assume, say, n < m. Divide n into m and get a remainder r1 < n. Divide r1 into n and get a remainder r2 < r1. Divide r2 into r1 and get a remainder r3 < r2. Keep doing this until the remainder is 0. The divisor r that generated this 0 remainder is gcd(n,m). This is because every common divisor of n,m divides r, and r is a common divisor of n,m.

The number of steps that the Euclidean algorithm takes is at most roughly the total number of digits in the problem - an enormous improvement over the "worst" algorithm.

There are also good candidates for the "worst" way to test whether or not a positive integer is prime. Just divide n by all of the numbers from 2 through n-1 and see if you find a factor.

For reference, there are approximately n/ln(n) primes below n. This is called the Prime Number Theorem.

Primality testing has gone through a very interesting evolution. The very best algorithms in practice has an unusual component which is not normally present. Namely, a probabilistic one.

In the Miller Rabin primality test, they cleverly associate, to any odd positive integer n, a set S[n] $\subset$ [1,n-1] with the following properties.

i. If n is prime then S[n] is empty.
ii. If n is composite then S[n] has at least (n-1)/2 elements.
iii. It is highly efficient to test for membership in S[n].

We can now "test" whether n is prime, highly efficiently. Generate a lot of numbers in [1,n-1], "randomly". Say a1,...,a200. Check for membership in S[n]. If at least one of them lies in S[n] then we know that n is composite. If none of them lie in S[n], then almost certainly n is prime.

Why? If it were composite then you would have a misleading outcome 200 times in a row! Chance of that is 2^{-200}.

This algorithm raises a number of deep foundational, philosophical, and metaphysical issues surrounding randomness and truth - in a particularly rich and focused way.

We now come to factoring. It is widely believed that factoring a positive integer is "hard" in the sense that it is going to take an unfeasible amount of time, generally, to factor positive integers randomly chosen with, say, hundreds of digits.

There has been a huge success in building up a theory and practice of cryptographic schemes based on the assumption that factoring positive integers chosen randomly is too difficult for anyone.

Nevertheless, there are methods that are considerably better than the "worst", which is to try everything. There is a family of methods called the sieve method, which originated in ancient times.

We now come to the solvability of equations. Let me concentrate on a single polynomial equation with integer coefficients. There are three key parameters: the number of variables, the degree, and the size of the coefficients.

1. There is an algorithm for testing solvability with real or complex number unknowns.

2. There is an algorithm for testing solvability for degree 2, with integer or rational or real or complex number unknowns.

3. There is no algorithm for testing solvability for 9 positive integer unknowns.

4. There is no algorithm for testing solvability for degree 4, with integer unknowns.

5. Nobody knows if there is an algorithm for testing solvability with rational unknowns.

6. Nobody knows if there is an algorithm for testing solvability for degree 3, with 2 rational unknowns. There is an algorithm for degree 3, with 2 integer unknowns.

There is, and has been, a huge amount of work trying to deal with the theoretical and practical issues surrounding the implementation of 1 and fragments, (first due to Alfred Tarski).

2. NO ALGORITHMS. Robust model of computation. Behavior of abstract machines, solvability of equations.

In order to establish that there is no algorithm for determining whether a property holds of inputs from a class, or no algorithm for returning information related to inputs from a class, one relies on a standard robust model of computation.

The first presentation of such a model was by Alan Turing. It was rather specialized, and considerable effort went into establishing its robustness.

Turing's formal computing devices were, on the surface, rather limited and restricted. So it was not entirely clear what would happen if one enlarged them without destroying the general features that they possessed that made them obviously algorithmic.

After much work along these lines, it became generally accepted that an extremely robust analysis had been given. In particular, the notions of partial recursive and recursive functions on nonnegative integers and on finite strings from a finite alphabet, as well as of recursively enumerable and recursive sets of nonnegative integers and finite strings from a finite alphabet, were fundamental, and "correctly analyze" clear informal notions.

In fact, this feeling was codified by Alonzo Church into what is called "Church's Thesis".

I am one of the very few people who has some optimism that there is some strikingly simple condition that can be placed on "computable" or "algorithm", which, in some way, "obviously" encompasses all "imaginable" computations and algorithms, and which is then proved to be equivalent to

the Turing model. Then we have an indisputable "proof" of Church's Thesis.

The first examples of algorithmically unsolvable problems were obtained by turning the models of computation on their heads.

Suppose there is an algorithm for testing whether a given Turing machine halts at a given input.

I.e., let TM do the following. Given any TM' and string x as input to TM, TM answers yes/no whether TM' halts with input x.

It is then easy to adjust TM to TM* that does the following. Given any TM' and string x as input to TM', TM* does not halt if TM' halts with input x, and TM* halts if TM' does not halt with input x.

By setting TM' = x = TM*, we get

With TM* and TM* as input to TM*, TM* does not halt if TM* halts with input TM*, and TM* halts if TM* does not halt with input TM*.

This is clearly impossible.

However theoretically satisfying this kind of thing might be to some, the down to earth mathematician is far more moved by their being a no algorithm result that is considerably closer to their mathematical interests.

There is a shortage of such negative results along these lines. I already mentioned the no algorithm results concerning solvability of polynomial equations in integer unknowns.

Another important one is that there is no algorithm for testing whether a given finitely generated group is trivial.

A related one in topology is: there is no algorithm for testing whether two closed 4-manifolds are homeomorphic.

3. SIMPLIFIED PHYSICAL SYSTEMS. Linear Order Systems. No algorithm for determining Boundedness. Research plans.

We now describe a very simplified kind of discrete physical system, based on finitely many bodies. We call these Linear Order Systems.

We prove that there is no way to analyze the behavior of such systems over eternity, in the following sense. There is no algorithm that determines, for any given initial configuration of the bodies, whether the evolution is contained in a finite region of space.

Moreover, the result is proved in the following strong form:

There is a specific Linear Order System with 12 bodies such that there is no algorithm that determines, for any given initial configuration of the 12 bodies, whether the evolution is contained in a finite region of space.

There has been considerable work on no algorithm results for simple machines. Below, we indicate how this work builds on existing work, and takes us into exciting new directions.

We are anxious to begin research in a number of directions. There are obvious extensions of Linear Order Systems to Planar Systems and Space Systems. We expect that the number of bodies required for the analogous results will be considerably smaller than 12.

Linear Order Systems involve quite general laws of motion, with few bodies. It will be important to extend the work to cover much more restricted laws of motion, perhaps with many more, but not too many more, bodies.

We now present the Linear Order Systems.

Space is identified with the integer number line

$$...-2,-1,0,1,2,...$$

We will not assume any reference point (e.g., 0).

Time is identified with the nonnegative integer number line

$$0,1,2,...$$

REMARK. An interesting direction of research is to allow arbitrary integers as times. We suspect that there are very interesting no algorithm results concerning what configurations can occur in a given Linear Order System.

In an n body Linear Order System, we have n bodies, $B1,...,Bn$, where $n \geq 1$. We write

$$Bi[t],\ 1 \leq i \leq n,$$

for the position of body Bi at time t. Each Bi[t] is an integer.

Thus the initial configuration of the n bodies is given by

$$B1[0],...,Bn[0].$$

Motion is deterministic, and very restricted. At each time > 0, each body will move to the left one unit, to the right one unit, or stay at the same position. I.e., we have

$$Bi[t+1] - Bi[t] \in \{-1,0,1\}.$$

In a Linear Order System, motion is completely specified in terms of the relative order of the n bodies.

The relative order of the n bodies $B1,...,Bn$ at time t is just the relative order of the n integers $B1[t],...,Bn[t]$. This is merely a tabulation of which of these integers is less than which others. Obviously, this information also tells us which of these integers is equaled to which others, and which of these integers is greater than which others.

There is a standard, compact way of specifying the relative order of $B1[t],...,Bn[t]$. It is by a listing

$$C1[t]\ ?\ C2[t]\ ?\ ...\ ?\ Cn[t]$$

where each ? is either $<$ or $=$, and $C1,...,Cn$ is a listing of the bodies $B1,...,Bn$. A useful convention is that if we see

$$Ci[t] = Cj[t]$$

adjacently in the listing, then we require i < j.

How can we justify this setup physically?

Think of each body as, e.g., constantly transmitting radiation with its unique signature. So each body is constantly aware of the presence of the other n-1 bodies, and knows which is which. In particular, each body is constantly aware of the presence of the other bodies to the limited extent of: is it to my left, to my right, or right here (at the same position)?

Furthermore, assume that each body can sense, given what it receives, the relative distance of each of the bodies from itself - at least to the extent of knowing which ones are closer or further away than others. E.g., by the strength of the received signals.

From this information, each body can obviously infer the relative order of ALL of the n bodies.

Each body then moves, by 0 or 1 units in each direction, according to this information only.

We are considering arbitrary rules of this kind, so that there is no assumption that different bodies may react in the same way to the same information.

Here is just one ***example of further research***: suppose each body only knows which bodies are to the left of it, to the right of it, and at the same position as it. This is a very restricted class of Order Systems. Then do we have a no algorithm theorem, and how many bodies do we need for such a no algorithm theorem?

But let us stick to what we know now at this first stage of development.

ONE BODY LINEAR ORDER SYSTEMS. There are exactly three one body Linear Order Systems. Always moving to the right, always moving to the left, always not moving. For any initial configuration, the first two are unbounded, and the third is bounded.

TWO BODY LINEAR ORDER SYSTEMS. There are three relative orders.

B1 = B2.
B1 < B2.
B2 < B1.

B1,B2 have to be assigned a number from -1,0,1 under each of the relative orders. There are 9 possibilities for this, and so we obtain 9^3 = 729 two body Linear Order Systems. It should be quite manageable - and interesting - to give a complete analysis of which of these laws lead to boundedness under which initial configurations, especially with the help of a computer to dispense with trivial cases, and to indicate what is true (that still needs to be proved), and cataloging everything.

THREE BODY LINEAR ORDER SYSTEMS. There are 13 relative orders.

B1 < B2 < B3.
B1 < B3 < B2.
B2 < B1 < B3.
B2 < B3 < B1.
B3 < B1 < B2.
B3 < B2 < B1
B1 = B2 < B3.
B1 = B3 < B2.
B2 = B3 < B1.
B1 < B2 = B3.
B2 < B1 = B3.
B3 < B1 = B2.
B1 = B2 = B3.

B1,B2,B3 have to be assigned a number from -1,0,1 under each of the relative orders. This consists of 39 numbers drawn from {-1,0,1}. Thus there are 27^13 = 3^39 three body Linear Order Systems. The challenge is to construct an efficient algorithm that determines boundedness for any three body Linear Order System under any given initial configuration of the three bodies. We conjecture that this can be done.

...

TWELVE BODY LINEAR ORDER SYSTEMS. Our theorem asserts that a boundedness test is impossible for at least one nine body Linear Order System.

Officially, any 12 body Linear Order System involves 12 numbers from {-1,0,1} assigned to each relative order for B1,....,B12. The number of such relative orders is rather large. E.g., considerably more than 12!. So each law of motion for 12 bodies officially requires a table of considerably more than 12(12!) numbers from -1,0,1.

However, there are many opportunities for providing far shorter descriptions of Linear Order Systems.

A partial relative order for n bodies, is a set of conditions of the form

$$Bi = Bj$$
$$Bi < Bj$$
$$Bi \square Bj$$
$$Bi \neq Bj$$

where $1 \square i,j \square n$. We can now write laws of motion in the following form.

Partial relative order #1. B1 moves ?,...,Bn moves ?
Partial relative order #2. B1 moves ?,...,Bn moves ?
...
Partial relative order #m. B1 moves ?,...,Bn moves ?
Otherwise. B1 moves ?,...,Bn moves ?.

We demand that these partial relative orders are mutually exclusive.

Let us call the total number of partial relative orders used, the presentation complexity.

Our intractable 12 body law of motion has presentation complexity far lower than the theoretical limit, which is greater than 12!, and probably can be massaged to have still lower presentation complexity – perhaps down to a few dozen.

QUESTION. What can we say about the triples (n,r) for which there is an n body Order System of presentation complexity at most r, such that boundedness for arbitrary initial configurations cannot be determined algorithmically?

Before getting into the details, we make some basic remarks on the relation between this work and previous work on abstract machines.

Interpretations of the no algorithms results on Turing machines, in terms of the motion of bodies, involves having arbitrarily large numbers of bodies, with no a priori bound. This is because they involve arbitrarily long strings of bits, or symbols from a finite alphabet.

Linear Order Systems are more closely related to register machines. But in Linear Order Systems, all abstract state and all flow of control must emanate from the configuration of the bodies alone - not from an ordered list of instructions and list of abstract states.

Of course, we rely heavily on previous work on Turing machines and (ideas from) register machines.

We anticipate a steady stream of restrictions on Linear Order Systems (such as the one mentioned earlier) and also the introduction of other models which take more and more aspects of actual physical systems into account.

The mathematics will then get increasingly deeper and more involved, getting further away from the earlier no algorithms work, which was not primarily motivated by this kind of direct consideration of physical systems.

4. SIMULATION OF TURING MACHINES.

We now show how to simulate the action of any Turing machine TM with two symbols and n states, by a Linear Order System, in the present sense, with considerably fewer than n bodies.

Because we are focused only on the determination of boundedness, we will make the assumption that the TM has no halting instructions. I.e., the TM is halting free. We will see that this requirement may cost at most one state.

Let the TM have symbols 0,1, and n states q1,...,qn. We use the standard quintuple formulation of TMs, with the two way infinite tape. We will use q1 as the initial state.

Inputs consist of two strings $x,y \in \{0,1\}^*$, of length ≥ 0, and a state qi. Initialization is by

...0001x qi y1000...

with the reading head on the first bit of y1, and the device in state q1.

TM operates by a complete set of 2n quintuples of the form

qi b c L qj
qi b c R qj
qi b - - -

where 1 ≤ i,j ≤ n, L = left, R = right, and b,c ∈ {0,1}. Here qi b - - - indicates halting when in state qi and reading bit b.

At any time, the TM is in the form

000...x q y000...

where x,y ∈ {0,1}* and q ∈ {q1,...,qn}. Here y is nonempty. The reading head is reading the first bit of y1.

In the simulation, we will use five bodies, B1,B2,B3 which are, however, augmented with some additional apparatus that is strictly forbidden in Linear Order Systems:

i. 0 as a reference point.
ii. Relative orders will use 0. I.e., they take the form 0 ? C1 ? C2 ? C3, where {C1,C2,C3} = {B1,B2,B3}.
iii. The n abstract states q1,...,qn.
iv. Two special states S1,S2.
v. Thus the "global states" will be the relative order (in the sense of ii) together with the q state (one of the n abstract states q1,...,qn), and the S1,S2 states.
vi. We assign -1,0,1 for movement, as usual.

We will then eliminate the use of the q,S1, and S2 states, thereby conforming to the Linear Order System model - with the exception that we will still be using the reference point 0. To accomplish this, we need to introduce ≥ log3(14n) bodies.

Finally, we can treat the reference point 0 as an additional body that does not move, resulting in a total of 4 + ⌈log3(14n)⌉ bodies.

For each state qi of TM, we define a law of motion M(qi), freely using partial global states. We will then put the M(qi) together in the obvious way.

M(qi) will depend on the unique quintuples in TM that start with qi. Let these be

qi 0 a □ qj
qi 1 b □ qk

It is convenient to split this into 16 cases, according to the values of a,b □ {0,1} and □,□ □ {L,R}.

We use B↑ for adding 1, and B□ for subtracting 1.

Initialization of M(qi) is with

B1 = the integer given by x.
B2 = the integer given by rev(y).
B3 = 0.
S1 = 0.
S2 = 0.
q = qi.

Here "rev" means reverse. x div 2 is the floor of x/2. x mod 2 is the parity of x (0 or 1).

Note that we have used S1,S2 instead of B4,B5. This is convenient, since we use S1,S2 like "states", where S1 always stays within [0,6], and S2 always stays within {0,1].

CASE 1. □ = □ = L. b = c = 0. We must transition to
B1 = x div 2.
B2 = 2y + (x mod 2) if y even; 2(y-1) + (x mod 2) if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qi if y even; qk if y odd.

1.1. S1 = 0, B3 < B2 □ B2□, B3↑.
1.2. S1 = 0, B2 = B3 □ S1 = 1.
1.3. S1 = 0, B2 < B3 □ S1 = 1, S2 = 1.
We now have S2 = y mod 2.
1.4. S1 = 1, B3 > 0 □ B2↑, B3□.
1.5. S1 = 1, S2 = 0, B3 = 0 □ S1 = 2.
1.6. S1 = 1, S2 = 1, B3 = 0 □ B2□, S1 = 2.
We now have B2 = y if y even; y-1 if y odd.
1.7. S1 = 2, B3 < B2 □ B3↑.
1.8. S1 = 2, B2 = B3 □ S1 = 3.

1.9. S1 = 3, B3 > 0 □ B2↑.
1.10. S1 = 3, B3 = 0 □ S1 = 4.
We now have B2 = 2y if y even; 2(y-1) if y odd.
1.11. S1 = 4, B3 < B1 □ B1↑, B3□.
1.12. S1 = 4, B1 = B3 □ S1 = 5.
1.13. S1 = 4, B1 < B3 □ B2↑, S1 = 5.
We now have B1 = x div 2, B2 = previous B2 if x is even; previous B2 plus 1 if x is odd.
1.14. S1 = 5, B3 > 0 □ B3□, S1 = 6.
1.15. S1 = 6, S2 = 0, B3 = 0 □ S1 = 0, q = qi.
1.16. S1 = 6, S2 = 1, B3 = 0 □ S1 = 0, S2 = 0, q = qk.

CASE 2. □ = □ = L. b = c = 1. We transition to
B1 = x div 2.
B2 = 2(y+1) + (x mod 2) if y even; 2y + (x mod 2) if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y mod 2 = 0; qk otherwise.

We have only to adjust 1.4 - 1.6 so that we have B2 = y+1 if y even; y if y odd.
2.4. S1 = 1, 0 < B3 □ B2↑, B3□.
2.5. S1 = 1, S2 = 0, B3 = 0 □ B2↑, S1 = 2.
2.6. S1 = 1, S2 = 1, B3 = 0 □ S1 = 2.

CASE 3. □ = □ = L. b = 0, c = 1. We transition to
B1 = x div 2.
B2 = 2y + (x mod 2).
B3 = 0.
S1 = 0.
q = qj if y mod 2 = 0; qk otherwise.

We have only to adjust 1.4 - 1.6 so that we have B2 = y.
3.4. S1 = 1, B3 > 0 □ B2↑, B3□.

CASE 4. □ = □ = L. b = 1, c = 0. We transition to
B1 = x div 2.
B2 = 2(y+1) + (x mod 2) if y even; 2(y-1) + (x mod 2) if y odd.
B3 = 0.
S1 = 0.
q = qj if y mod 2 = 0; qk otherwise.

We have only to adjust 1.4 - 1.6 so that we have B2 = y+1 if y even; y-1 if y odd.
4.4. S1 = 1, 0 < B3 □ B2↑, B3□.

4.5. S1 = 1, S2 = 0, B3 = 0 □ B2↑, S1 = 2.
4.6. S1 = 1, S2 = 1, B3 = 0 □ B2□, S1 = 2.

CASE 5. □ = □ = R. b = c = 0. We transition to
B1 = 2x.
B2 = y div 2.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y even; qk if y odd.

5.1. S1 = 0, B3 < B2 □ B2□, B3↑.
5.2. S1 = 0, B2 = B3 □ S1 = 1.
5.3. S1 = 0, B2 < B3 □ S1 = 1, S2 = 1.
We now have B2 = y div 2, S2 = y mod 2.
5.4. S1 = 1, 0 < B3 □ B3□.
5.5. S1 = 1, B3 = 0 □ S1 = 2.
5.6. S1 = 2, B3 < B1 □ B3↑.
5.7. S1 = 2, B1 = B3 □ S1 = 3.
5.8. S1 = 3, B3 > 0 □ B1↑, B3□.
We now have B1 = 2x.
5.9. S1 = 3, S2 = 0, B3 = 0 □ S1 = 0, q = qj.
5.10. S1 = 3, S2 = 1, B3 = 0 □ S1 = 0, S2 = 0, q = qk.

CASE 6. □ = □ = R. b = c = 1. We transition to
B1 = 2x+1.
B2 = y div 2.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y mod 2 = 0; qk otherwise.

We have only to adjust 5.9, 5.10 so that we have B1 = 2x+1.

6.9. S1 = 3, S2 = 0, B3 = 0 □ B1↑, S1 = 0, q = qj.
6.10. S1 = 3, S2 = 1, B3 = 0 □ B1↑, S1 = 0, S2 = 0, q = qk.

CASE 7. □ = □ = R. b = 0, c = 1. We transition to
B1 = 2x if y even; 2x+1 if y odd.
B2 = y div 2.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y mod 2 = 0; qk otherwise.

We have only to adjust 5.10 so that we have B1 = 2x if y even; 2x+1 if y odd.

7.10. S1 = 3, B2 = 1, B3 = 0 □ B1↑, S1 = 0, S2 = 0, q = qk.

CASE 8. □ = □ = R. b = 1, c = 0. We transition to
B1 = 2x+1 if y even; 2x if y odd.
B2 = y div 2.
B3 = 0.
S1 = 0.
S2 = 0.
q = qi if y mod 2 = 0; qk otherwise.

We have only to adjust 5.9 so that we have B1 = 2x+1 if y even; 2x if y odd.

8.9. S1 = 3, S2 = 0, B3 = 0 □ B1↑, S1 = 0, q = qj.

CASE 9. □ = L, □ = R, b = c = 0. We transition to
B1 = x div 2 if y even; 2x if y odd.
B2 = 2y + (x mod 2) if y even; y div 2 if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y even; qk if y odd.

9.1 S1 = 0, B3 < B2 □ B2□, B3↑.
9.2. S1 = 0, B2 = B3 □ S1 = 1.
9.3. S1 = 0, B2 < B3 □ S1 = 1, S2 = 1.
We now have S2 = y mod 2, B2 = y div 2.

We now list clauses pertaining to y even; i.e., S2 = 0.
Note that we can repeat Case 1 with S2 = 0, after 1.3.

9.4. S1 = 1, S2 = 0, B3 > 0 □ B2↑, B3□.
9.5. S1 = 1, S2 = 0, B3 = 0 □ S1 = 2.
We now have B2 = y.
9.6. S1 = 2, S2 = 0, B3 < B2 □ B3↑.
9.7. S1 = 2, S2 = 0, B2 = B3 □ S1 = 3.
9.8. S1 = 3, S2 = 0, B3 > 0 □ B2↑.
9.9. S1 = 3, S2 = 0, B3 = 0 □ S1 = 4.
We now have B2 = 2y.
9.10. S1 = 4, S2 = 0, B3 < B1 □ B1↑, B3□.
9.11. S1 = 4, S2 = 0, B1 = B3 □ S1 = 5.
9.12. S1 = 4, S2 = 0, B1 < B3 □ B2↑, S1 = 5.
We now have B1 = x div 2, B2 = 2y + x mod 2.
9.13. S1 = 5, S2 = 0, B3 > 0 □ B3□, S1 = 6.
9.14. S1 = 6, S2 = 0, B3 = 0 □ S1 = 0, q = qj.
This is final for y even.

We now list clauses pertaining y odd; i.e., S2 = 1. Note that we can repeat Case 5 with S1 = 1, after 5.3.

9.15. S1 = 1, S2 = 1, 0 < B3 □ B3□.
9.16. S1 = 1, S2 = 1, B3 = 0 □ S1 = 2.
9.17. S1 = 2, S2 = 1, B3 < B1 □ B3↑.
9.18. S1 = 2, S2 = 1, B1 = B3 □ S1 = 3.
9.19. S1 = 3, S2 = 1, B3 > 0 □ B1↑, B3□.
We now have B1 = 2x.
9.20. S1 = 3, S2 = 1, B3 = 0 □ S1 = 0, S2 = 0, q = qk.
This is final for y odd.

CASE 10. □ = L, □ = R, b = c = 1. We transition to
B1 = x div 2 if y even; 2x+1 if y odd.
B2 = 2(y+1) + (x mod 2) if y even; y div 2 if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y even; qk if y odd.

10.1 S1 = 0, B3 < B2 □ B2□, B3↑.
10.2. S1 = 0, B2 = B3 □ S1 = 1.
10.3. S1 = 0, B2 < B3 □ S1 = 1, S2 = 1.
We now have S2 = y mod 2, B2 = y div 2.

We have only to adjust 9.20 so that B1 = 2x+1, and 9.5 so that B2 = y+1.

10.5. S1 = 1, S2 = 0, B3 = 0 □ B1↑, S1 = 2.
10.20. S1 = 3, S2 = 1, B3 = 0 □ B1↑, S1 = 0, S2 = 0, q = qk.

CASE 11. □ = L, □ = R, b = 0, c = 1. We transition to
B1 = x div 2 if y even; 2x+1 if y odd.
B2 = 2y + (x mod 2) if y even; y div 2 if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y even; qk if y odd.

11.1 S1 = 0, B3 < B2 □ B2□, B3↑.
11.2. S1 = 0, B2 = B3 □ S1 = 1.
11.3. S1 = 0, B2 < B3 □ S1 = 1, S2 = 1.
We now have S2 = y mod 2, B2 = y div 2.

We have only to adjust 9.20 so that B1 = 2x+1.

11.20. S1 = 3, S2 = 1, B3 = 0 □ B1↑, S1 = 0, S2 = 0, q = qk.

CASE 12. □ = L, □ = R, b = 1, c = 0. We transition to
B1 = x div 2 if y even; 2x if y odd.
B2 = 2(y+1) + (x mod 2) if y even; y div 2 if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y even; qk if y odd.

12.1 S1 = 0, B3 < B2 □ B2□, B3↑.
12.2. S1 = 0, B2 = B3 □ S1 = 1.
12.3. S1 = 0, B2 < B3 □ S1 = 1, S2 = 1.
We now have S2 = y mod 2, B2 = y div 2.

We have only to adjust 9.5 so that B2 = y+1.

12.5. S1 = 1, S2 = 0, B3 = 0 □ B2↑, S1 = 2.

CASE 13. □ = R, □ = L. b = c = 0. We transition to
B1 = 2x if y even; x div 2 if y odd.
B2 = y div 2 if y even; 2(y-1) + (x mod 2) if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y even; qk if y odd.

13.1 S1 = 0, B3 < B2 □ B2□, B3↑.
13.2. S1 = 0, B2 = B3 □ S1 = 1.
13.3. S1 = 0, B2 < B3 □ S1 = 1, S2 = 1.
We now have S2 = y mod 2, B2 = y div 2.

We now list clauses pertaining to y even; i.e., S2 = 0.
Note that we can repeat Case 5 with S2 = 0, after 5.3.

13.4. S1 = 1, S2 = 0, 0 < B3 □ B3□.
13.5. S1 = 1, S2 = 0, B3 = 0 □ S1 = 2.
13.6. S1 = 2, S2 = 0, B3 < B1 □ B3↑.
13.7. S1 = 2, S2 = 0, B1 = B3 □ S1 = 3.
13.8. S1 = 3, S2 = 0, B3 > 0 □ B1↑, B3□.
We now have B1 = 2x.
13.9. S1 = 3, S2 = 0, B3 = 0 □ S1 = 0, q = qj.
This is final for y even.

We now list clauses pertaining to y odd; i.e.., S2 = 1.
Note that we can repeat Case 1 for S2 = 1, after 1.3.

13.10. S1 = 1, S2 = 1, B3 > 0 → B2↑, B3↓.
13.11. S1 = 1, S2 = 1, B3 = 0 → B2↓, S1 = 2.
We now have B2 = y-1.
13.12. S1 = 2, S2 = 1, B3 < B2 → B3↑.
13.13. S1 = 2, S2 = 1, B2 = B3 → S1 = 3.
13.14. S1 = 3, S2 = 1, B3 > 0 → B2↑.
13.15. S1 = 3, S2 = 1, B3 = 0 → S1 = 4.
We now have B2 = 2(y-1). Also B3 = 0.
13.16. S1 = 4, S2 = 1, B3 < B1 → B1↑, B3↓.
13.17. S1 = 4, S2 = 1, B1 = B3 → S1 = 5.
13.18. S1 = 4, S2 = 1, B1 < B3 → B2↑, S1 = 5.
We now have B1 = x div 2, B2 = previous B2 plus 1.
13.19. S1 = 5, S2 = 1, B3 > 0 → B3↓, S1 = 6.
13.20. S1 = 6, S2 = 1, B3 = 0 → S1 = 0, S2 = 0, q = qk.
This is final for y odd.

CASE 14. → = R, ← = L. b = c = 1. We transition to
B1 = 2x+1 if y even; x div 2 if y odd.
B2 = y div 2 if y even; 2y + (x mod 2) if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y even; qk if y odd.

14.1 S1 = 0, B3 < B2 → B2↓, B3↑.
14.2. S1 = 0, B2 = B3 → S1 = 1.
14.3. S1 = 0, B2 < B3 → S1 = 1, S2 = 1.
We now have S2 = y mod 2, B2 = y div 2.

We have only to adjust clause 13.9 so that B1 = 2x+1, and clause 13.11 so that B2 = y.

14.9. S1 = 3, S2 = 0, B3 = 0 → B1↑, S1 = 0, q = qj.
14.11. S1 = 1, S2 = 1, B3 = 0 → S1 = 2.

CASE 15. → = R, ← = L. b = 0, c = 1. We transition to
B1 = 2x if y even; x div 2 if y odd.
B2 = y div 2 if y even; 2y + (x mod 2) if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y even; qk if y odd.

15.1 S1 = 0, B3 < B2 → B2↓, B3↑.

15.2. S1 = 0, B2 = B3 → S1 = 1.
15.3. S1 = 0, B2 < B3 → S1 = 1, S2 = 1.
We now have S2 = y mod 2, B2 = y div 2.

We have only to adjust clause 13.11 so that B2 = y.

15.11. S1 = 1, S2 = 1, B3 = 0 → S1 = 2.

CASE 16. □ = R, □ = L. b = 1, c = 0. We transition to
B1 = 2x+1 if y even; x div 2 if y odd.
B2 = y div 2 if y even; 2(y-1) + (x mod 2) if y odd.
B3 = 0.
S1 = 0.
S2 = 0.
q = qj if y even; qk if y odd.

16.1 S1 = 0, B3 < B2 → B2□, B3↑.
16.2. S1 = 0, B2 = B3 → S1 = 1.
16.3. S1 = 0, B2 < B3 → S1 = 1, S2 = 1.
We now have S2 = y mod 2, B2 = y div 2.

We have only to adjust clause 13.9 so that B1 = 2x+1.

16.9. S1 = 3, S2 = 0, B3 = 0 → B1↑, S1 = 0, q = qj.

This completes the description of M(qi).

Note that in each of the 16 cases, S1 stays in [0,6] and S2 stays in [0,1]. Hence there is a total of most 14 states of S1,S2 in each M(qi). So there is a total of at most 14n auxiliary states.

A more careful analysis shows that in each M(qi), there are at most 11 states of S1,S2. Hence there is a total of at most 11n auxiliary states.

Suppose log2(m) ≥ 11n. Then we can replace the auxiliary states with m new bodies which occupy positions 0,1 only. We can transition in any way from one of these to any other.

Hence we have accomplished the simulation of TM with 3 + log2(m) bodies, using 0 as a reference point. We can view the reference point as another body that does not move. Hence we have accomplished the simulation with a 4 + log2(m) body Linear Order System.

We can now prove the following.

THEOREM 4.1. Suppose there is a TM with 2 symbols and n states, for which the halting problem is algorithmically unsolvable. Suppose m ≥ log(11(n+1)). Then there is a 4+m body Linear Order System LOS whose boundedness problem is algorithmically unsolvable. If the former is complete r.e. then the latter is complete r.e.

Proof: Let TM be as given. Let TM' be obtained from TM by replacing each halting quintuple qi b HALT in TM by the quintuples

qi b b L qn+1
qn+1 0 0 R qi
qn+1 1 1 R qi

Obviously, any run of TM that doesn't halt is a run of TM'. Now consider any run of TM that does halt, say with qi b HALT.

Then TM' will also arrive at qi b, and execute the following instructions indefinitely in a loop:

qi b b L qn+1
qn+1 c c R qi
qi b b L qn+1

Hence the corresponding run of TM' is bounded.

Suppose that boundedness for TM' is algorithmically decidable. We now algorithmically solve the halting problem for TM.

Let X be the run of TM at input □. Let X* be the run of TM' at input □.

case 1. X* is unbounded. Then X cannot halt, for then X* would be bounded. We have solved the halting problem for TM in this case.

case 2. X* is bounded. Then X* goes into a loop, and we can examine the quintuples executed in X* to see if any of them are qi b, where qi b HALT is in TM. It is obvious that X halts if and only if we find such.

This solves the halting problem for TM, contrary to the hypothesis. Therefore boundedness for TM' is algorithmically undecidable.

Moreover, if the halting problem for TM is complete r.e., then boundedness for TM' is complete r.e., by the same argument.

We now let m be as given. I.e., the number of order types of m tuples is at most 14(n+1). We let LOS be the 4+m body Linear Order System that simulates TM' at any input, as given above. Then boundedness of TM' is equivalent to boundedness of LOS. QED

LEMMA 4.2. There is a TM with 2 symbols and 19 states whose halting problem is complete r.e.

Proof: See

Baiocchi: Three Small Universal Turing Machines. In: Springer
LNCS 2055, Machines, Computation, and Universality, 2001, pp. 1-10.

QED

THEOREM 4.3. There is a 12 body Linear Order System such that there is no algorithm for determining whether an arbitrary initial configuration has bounded evolution. In fact, the boundedness problem is complete r.e.

Proof: If we use Theorem 4.1 with 11(n+1), then we need $2^8 = 128 \geq 11(20) = 220$. QED

5. AXIOMS FOR MATHEMATICS. The ZFC axioms.

For the next section, we need to be aware that already by the early part of the 20th century, the standard axiomatic basis for mathematics had been carefully formulated and reasonably well accepted.

This is the so called ZFC axioms, or Zermelo Frankel set theory with the axiom of choice.

The vast preponderance of mathematics can, without difficulty, be proved within ZFC.

Notable exceptions include various unusually and heavily set theoretic problems, plus some examples of ours that are much more down to earth, and will be discussed in the soon forthcoming book

Boolean Relation Theory and Incompleteness Phenomena.

6. INDIVIDUAL SENTENCES NOT DECIDED IN ZFC. Consequence of no algorithms. Research plans.

There is a generally much more delicate, deeper, and more difficult kind of "undecidability", that is often confused with algorithmic undecidability.

This concerns a **SINGLE STATEMENT**. Not an infinite family of statements.

There is no clear way to get at the hardness of a SINGLE QUESTION using algorithms. Obviously, there is an algorithm for deciding whether a single given statement S is true or not.

case 1. S is true. Let ALG be the algorithm that always says true. Then ALG correctly decides the truth value of S.

case 2. S is false. Let ALG be the algorithm that always says false. Then ALG correctly decides the truth value of S.

So how do we talk about "undecidability" in the context of a single statement S?

One simple way is to assert that it is not provable or refutable in ZFC.

From the early by now standard work in mathematical logic, we have the following.

THEOREM. Let S be a set of positive integers defined in ZFC. There exists a positive integer n such that the statement "n □ S" is neither provable nor refutable in ZFC.

With a little bit of fiddling, this tells us that there is a 12 body Order System with initial conditions, for which

the statement “is it bounded” is neither provable nor refutable in ZFC.

But this misses the essential point. If we pass through this classic Theorem, the number of digits needed to present the initial condition is grotesque.

On the other hand, if we require that the bodies start out at the same place, then the number of bodies needed becomes grotesque.

CONJECTURE

There is a 20 body Linear Order System such that ZFC neither proves nor refutes boundedness with the initial configuration having all bodies at the same point.

As remarked before, an arbitrary 20 body Linear Order System may have so little in the way of symmetry that it may be unfeasible to describe it.

Here is a much strong conjecture.

There is a 20 body Linear Order System of presentation complexity at most 50 such that ZFC neither proves nor refutes boundedness with the initial configuration having all bodies at the same point.

In a discussion yesterday with Professor Inwagen, he asked me what I could hope to say about the situation with exactly two bodies.

I saw right away that this might well turn out to be an extremely fruitful path to follow.

We can very naturally allow the motion of each body to depend on the exact distance (positive or negative or 0) to the other, up to magnitude, say, 100. For signed distances of magnitude greater than 100, we would require a uniform choice of motion.

If there proves to be insufficient “juice” to get these kinds of negative results, then we can allow the motion of each body to be given by any number from -100 to 100.

The conjecture would be that we get algorithmic undecidability with general initial position, and we get

ZFC undecidability with initial position having both particles at the same position.

At the other extreme, physicists may be more attracted to formulations where the number of particles is fixed but quite large, spread out with a realistic initial configuration, but where the “laws of motion” are extremely simple – and the same for each body. Each body has a position and also one of a very few states, which determine how it interacts with other bodies in its vicinity.

It is very likely that this modest initial work can be crafted to apply well to such systems.