On The Halting Problem

Louis H. Kaufffman
University of Illinois at Chicago, Mathematics Department
851 South Morgan Street, Chicago, IL 60607-7045

Abstract
This paper discusses the presence of undecidable problems in mathematics, with reference to the halting problem for algorithms, the hypergame paradox and the paradox of the well-founded sets. All these results are curiously related to self-reference and the strange loop by which a self-observing system can indicate its own operations to itself.
Keywords: algorithm, halting problem, undecidable problem, hypergame paradox, well-founded set, sell-reference, strange loop, self-observing system

1 Introduction

Some mathematical problems are not decidable.

This came as a shock to some mathematicians who had hoped that any well-stated problem would have a solution and that we only need persevere to find that solution. Sometimes this attitude has been right. A good example is the famous Last Theorem of Fermat that was eventually proved 200 years after Fermat by Andrew Wiles, using all the resources of twentieth century mathematics. There are some problems that are unlikely to see a solution but we do not know if they are truly undecidable.

For example, recall the sequence of prime numbers

2,3,5,7,11,13,17,19,23,29,31,37,41,43,...

Each prime number is divisible only by itself and the number 1.

One can ask whether there are infinitely many prime numbers in the Fibonacci Series

1,1,2,3,,5,8,13,21,34,55,89,144,289,...

The Fibonacci Series has been known since the 1500's and the question of primes in that series has surely been investigated since that time. Euclid proved in 500BC that there are infinitely many prime numbers, but the same question for the Fibonacci Series remains open. Is this an undecideable problem? We do not know, but most mathematicians would take the question as a challenge and try to solve it, if they care to put their mind to it.

I can give you a problem that is inherently unsolvable by giving you contradictory instructions, but we assume that this avenue is not available to you. We assume that the mathematical question and the system in which it is asked is consistent. It will not do to ask you for the truth value of the sentence. "This sentence is false." We will refer to such unsolvable problems as directly unsolvable problems. But maybe all undecidable problems are direct. Could it be? We shall attempt to find out.

We shall find out that directly undecidable problems may not be so direct.

They will show us the limitations of our constructions and conceptions. Here is an example.

International Journal of Computing Anticipatory Systems, Volume 28, 2014
Edited by D. M. Dubois, CHAOS, Liège, Belgium, ISSN 1373-5411 ISBN 2-930396-17-2

1.1 The Hypergame Paradox

Let's play a game. I call it "hypergame". To play this game, the first player says "Let's play hypergame." Then the second player says "Let's play X." where X is the name of any game that ends in a finite amount of time. The two players then play one round of X just as one usually does play X. For example, first player says "Let's play hypergame." and second player says "Let's play tennis.", and then they go out and play a round of tennis.

We will call a game finite if it ends in a finite amount of time whenever it is played. As you can see, *hypergame is a finite game*, since the second player always chooses a finite game. But then, according to the rules of hypergame we could have the following exchange: A. Let's play hypergame. B. Let's play hypergame. A. Let's play hypergame. B. Let's play hypergame. ... This is a round of hypergame, using the fact that hypergame is a finite game and so can itself be chosen. But this is absurd! We have shown a round of hypergame that never ends! If hypergame is a finite game, then hypergame is NOT a finite game. What should we do with this paradox? Think about this while you read the rest of the paper.

1.2 The Sorcerer's Apprentice

We'll mention one more fable - The Sorcerer's Apprentice. This story was made famous by a Disney cartoon starring Mickey Mouse as the apprentice. According to the Wikipedia [5]:

"*The Sorcerer's Apprentice* is the English name of a poem by Goethe, *Der Zauberldhrling,* written in 1797. The poem begins as an old sorcerer departs his workshop, leaving his apprentice with chores to perform. Tired of fetching water by pail, the apprentice enchants a broom to do the work for him -- using magic he is not yet fully trained in. The floor is soon awash with water, and the apprentice realizes that he cannot stop the broom because he does not know how. Not knowing how to control the enchanted broom, the apprentice splits it in two with an axe, but each of the pieces becomes a new broom and takes up a pail and continues fetching water, now at twice the speed. When all seems lost, the old sorcerer returns, quickly breaks the spell and saves the day. The poem finishes with the old sorcerer's statement that powerful spirits should only be called by the master himself."

The theme of this story haunts the mathematical tale we are about to tell. In our tale the sorcerer's apprentice is the machine, or logical formal language that must follow only its own rules. Such languages can get stuck and multiply themselves and never halt. Perhaps the sorcerer is the mathematician or logician who can understand, and stop the runaway machine or at least understand that the machine has gone into an infinite loop. But the real sorcerer would always know when the runaway condition was upon us and would be able to avert it. Here we prove that a limited machine can never be a sorcerer. But then where is the sorcerer? Is there really an oracle who can save us, poor apprentices? We do not answer this question here. The question of the existence of the sorcerer to whom we may be apprenticed is a central conundrum that is now enfolded into second order cybernetics. Second order cybernetics studies observing systems,

including those that observe themselves. Are there any such systems? Or do we live in a world of apprentices, all with the illusion that they are each autonomous sorcerers.

2 The Halting Problem

If you are presented with a specific set of instructions, you may be able to tell if following them will go on forever or whether the process will stop. For example, consider the following instructions:

2.1 Golden Calculation Algorithm

1. Let **P = 1**.
2. Replace **P** by **1 + 1/P**.
3. If the absolute value of $\mathbf{P^2 - P - 1}$ is less than **1/1000000** then print the value of **P** and **STOP**. Otherwise, go to step 2.

These three instructions form an algorithm to compute the Golden Ratio **G**,

$$\mathbf{G = (1 + Sqrt(5))/2,}$$

to an accuracy of one millionth. A mathematician with the usual skills can analyse this algorithm and show that it will stop.

On the other hand consider this algorithm.

1. Go to step 2.
2. Go to step 1.

This algorithm does not halt. Anyone or any machine following these instructions just shuttles back and forth between step 1 and step 2.

Now consider the following algorithm. We'll call this one the

2.2 Collatz Algorithm

1. Choose a positive integer **N**.
2. If **N** is even, replace **N** by **N/2**. If **N** is even go to step 2, otherwise go to step 3.
3. If **N** is odd, replace **N** by **3N + 1**. Go to step 4.
4. If **N = 1**, **STOP**. Otherwise go to step 2.

Here we assume that if some stated condition is not met, then the operator of the algorithm does not perform the instruction. Thus in step 2, if the number **N** is odd, the operator proceeds to step 3. Note the epistemology. An algorithm has an operator, either a person or a machine, who follows the instructions exactly. The operator follows instructions "to the letter" in the idiom of standard English.

We can summarize the Collatz Algorithm by:

Choose a number.
If it is even, divide it by 2.
If it is odd, multiply it by 3 and add 1.
Keep doing this until you get the number 1.

For example you can start with **N = 7**.

Then the process obtained by following the instructions in the Collatz Algorithm goes as follows:

7			
3 x 7 + 1 = 22	**3 x 17 + 1 = 52**	**40/2 = 20**	**16/2 = 8**
22/2 = 11	**52/2 = 26**	**20/2 = 10**	**8/2 = 4**
3 x 11 + 1 = 34	**26/2 = 13**	**10/2 = 5**	**4/2 = 2**
34/2 = 17	**3 x 13 + 1 = 40**	**3 x 5 + 1 = 16**	**2/2 = 1**
			STOP

This algorithm has been examined experimentally, and it appears to always stop for every input **N**. But *no one has found a proof that this algorithm halts for every N.* We just do not know.

We have algorithms that are known to halt, algorithms that do not halt, and algorithms where we do not (yet) know if they halt.

The halting question is most interesting when the algorithm can have infinitely many inputs as in the Collatz Algorithm. Of course there are algorithms with only one input that are still worth analyzing. For example consider the following question: *Does **Pi** have a consecutive string of one million identical digits?* A single program can investigate this for us by always getting one more digit of **Pi** (by a computation) and then looking to see if the new digit is part of a string of one million consecutive digits. The program will halt if it finds such a string and keep on running otherwise.

This is similar to a game we have all played: Can I toss a coin and get many consecutive heads. I will start tossing and stop as soon as I get a tail. This algorithm always halts for a human being tossing a coin, but in principle it does not have to halt!

Return to the **Pi** question. If we set a computer to investigating this question, the computer will halt if there is a consecutive sequence of a million consecutive digits. But if it has not halted for the last 10 years, this means nothing. So we wait and we wait and maybe it will never halt. We need a theory to decide it faster, but maybe there is no such theory. Some questions can only be answered by investigating them directly.

3 Is there a Mathematical Oracle?

If there were a machine that could tell whether any algorithm that you gave it would halt, such a machine would be a mathematical oracle of a very high order. It is natural for us to ask whether such a wonderful machine could possibly exist!

We shall see that it is *not possible* for such a machine to exist if we assume that all the algorithms that it is given are written in its own terms. This means that if there were an oracle, it would have to be at a different level than all the algorithms that it would analyze.

First I want to show you the core of this proof. I will show that the set of algorithms is inexhaustible! *If you give me an infinite list of halting algorithms, I will produce a new halting algorithm that cannot possibly be on the list.*

To see how to do this, we will need some notation.

Let

$$\mathbf{L = \{A_1, A_2, A_3, ...\}}$$

be an infinite list of halting algorithms. We assume that each $\mathbf{A_k}$ is an algorithm that halts whenever you give it one of the numbers **1,2,3, 4,....**

We let $\mathbf{A_k(N)}$ denote the behaviour of the algorithm $\mathbf{A_k}$ when you give it input **N**. The only thing we officially know about $\mathbf{A_k(N)}$ is that this behaviour stops after a finite number of steps, because we have assumed that $\mathbf{A_k}$ does halt for every input **N**.

Now I will tell you a new algorithm **M.**

Note that I do not give **M** a subscript.

I will show that **M** *cannot* be on the list **L.**

I call **M** the "Monkey Wrench" for the list **L.**

Here is **M's** specification.

3.1 The Monkey Wrench Algorithm

1. Input **N.**
2. Go to the list **L** and extract the algorithm $\mathbf{A_N}$.
3. Apply $\mathbf{A_N}$ to **N** (finding the behaviour $\mathbf{A_N(N)}$).
4. Stop when $\mathbf{A_N(N)}$ stops.

Note that the Monkey Wrench always halts no matter what number **N** you give it, for when you give **M** the number **N**, **M** computes $\mathbf{A_N(N)}$ and we know that $\mathbf{A_N}$ halts for any input to $\mathbf{A_N}$ and for any **N.** Thus Monkey Wrench is a halting algorithm and it is *possible* that Monkey Wrench might be on our list **L.**

Theorem. *The Monkey Wrench M for the list L is not on the list L.*

Proof. Suppose that $\mathbf{M = A_N}$ for some **N.**

For the sake of our argument lets say $\mathbf{M = A_{137}}$.

So let's see how this would work.

Suppose we want to find out **M(1).** The procedure (above) tells us to find $\mathbf{A_1}$ and to compute $\mathbf{A_1(1)}$ and this will certainly work out and halt since we have been given $\mathbf{A_1}$ all along.

But now suppose we try **M(137).** Then we "Go to the list **L** and extract the algorithm $\mathbf{A_{137}}$." But we have assumed that $\mathbf{A_{137} = M}$. Therefore when we look at the list and find the text for $\mathbf{A_{137}}$ it reads

TEXT FOR ALGORITHM $\mathbf{A_{137}}$.

1. Input **N.**
2. Go to the list **L** and extract the algorithm $\mathbf{A_N}$.
3. Apply $\mathbf{A_N}$ to **N** (finding the behaviour $\mathbf{A_N(N)}$).
4. Stop when $\mathbf{A_N(N)}$ stops.

$\mathbf{A_{137}}$ has this text because we have assumed that $\mathbf{A_{137} = M}$. The text is the text for **M.**

We are supposed to apply $\mathbf{A_{137}(137)}$. So we use the input **N = 137.** Step 2 says "Go to the list **L** and extract algorithm $\mathbf{A_{137}}$." But this is just where we started. So all our algorithm can do when asked to find **M(137)** is to keep looking up the entry under $\mathbf{A_{137}}$ again and again. **M(137)** does not halt! It has an infinite loop when you give it the input **N = 137.**

We have shown that if **M** = **Ak** for any specific **k**, then **M(k)** will not halt. (There was nothing special about 137. We could have used any number at all.) Since **M** *does* halt for every **k**, we conclude that **M** is not on the list **L** of halting algorithms. **Q.E.D.**

We have proved that there is no such thing as a complete list of all halting algorithms. As soon as we produce any infinite list of halting algorithms, we can use that very list to make a new algorithm that is not on the list.

It is this richness in the set of algorithms that makes the halting problem undecidable when we restrict the whole situation to a given language, call it Losp. If you work in a given language, then it is possible to make a list of all the algorithms in that language using a program in the language. That is, there would be a Losp program that will find the k-th algorithm if you give the program the number k. This is because we can use Losp to recognize when some list of instructions is a grammatically valid program in Losp. A grammatically correct Losp algorithm does not have to halt, but we would like to find all the ones that halt.

If there were some way to decide whether or not each algorithm on that list of all algorithms halted, we could make a *smaller* list of exactly those algorithms (in Losp) that halt. If the method for deciding whether an algorithm halts were *itself* a program in the language, then we would find that we could write down the Monkey Wrench algorithm in the language as well. This would lead immediately to a contradiction, since the Monkey Wrench would produce a halting algorithm not on the purportedly complete list. I will give a few more details below.

How do we know that the Monkey Wrench could be written in the given programming language (Losp)? Well if the program **D** that decides whether an algorithm halts is written in Losp, then we can write the text for **M** as follows.

3.2 The Monkey Wrench for Losp

1. Input **N**.
2. Form an empty list **L**.
3. List the next algorithm in Losp.
   If no algorithms have been listed, list the first algorithm in Losp.
4. Test the algorithm **A** just produced using **D**.
   If **A** passes the halting test, place it on the list.
   If **A** passes and is the **N**-th algorithm on the list, then go to 5. Otherwise go to 3.
5. Run **A(N)**.

This more technical specification of the Monkey Wrench **M** does that same thing that our old one did. It runs the **N**-th halting algorithm with the number **N** as input. By our previous argument, we know that **M** cannot be on the supposedly complete list of halting algorithms in Losp. This is a the contradiction inherent in assuming that it is possible to decide the halting problem in Losp with a Losp algorithm. We cannot get Losp to decide the halting problem about its own algorithms.

4 Comment on the Undecidability of the Halting Problem

The logic of our argument about the halting problem is essentially the same as the following. A set **X** is well-founded if **X** has no infinite descending chains of

membership. Note that a set that is a member of itself cannot be well-founded. The simplest example would (formally) be

$$S = \{S\}.$$

We then have

$$S = \{S\} = \{\{S\}\} = \{\{\{S\}\}\} = ...$$

forming an infinite chain of members. **S** is a member of itself, so **S** is a member of a member of itself, so **S** is a member of a member of a member of itself and so on ad infinitum.

In the domain of concepts there are collections that are members of themselves. For example the collection of ideas is itself an idea. No set that is a member of itself is well-founded.

Theorem. Any set of well founded sets is incomplete. In particular, if **W** is a set of well founded sets, then **W** is itself well-founded and **W** is not a member of itself.

Proof. Any set **W** of well-founded sets is itself well founded. *For if we look for a descending chain of membership in **W**, we shall have to choose a member of **W** and look there.* But that member is well- founded and so all chains of membership in that element terminate. **W** cannot be a member of itself, for then **W** would have an infinite descending chain of membership. **Q.E.D.**

This Theorem tells us that the collection of all well-founded sets cannot be a set. For if **AW** is the collection of all well-founded sets, and **AW** is itself a set, then we know that **AW** is well-founded, but then since **AW** is ALL well-founded sets, we should have **AW** a member of itself. This leads to a contradiction. This is sometimes called the Well-Founded Set Paradox. It is usually solved by declaring that the collection of all well-founded sets is not itself a set. This result about well-founded sets has just the same logical structure as our discussion of the hypergame paradox in the introduction to the paper. In that paradox we defined hypergame as a kind of collection of all finite games. But hypergame as a game then becomes a finite game. The problem is that we cannot make hypergame one of the games that hypergame uses without falling down the rabbit hole. The structure of the paradox is the same. But for games it seems very strange to have to declare hypergame not quite a game. The pattern of this argument about the incompleteness of well-formed sets is the same as our construction of the Monkey Wrench algorithm **W**. Halting is analogous to being well-founded.

5 Commentary

We imagined a big, general-purpose computer that can run all sorts of algorithms in a single language in which it is programmed. Then we asked whether there is a program for this machine that can take any algorithm written in the machine's language and compute, in a finite amount of time, whether or not that algorithm will always halt. We showed that this cannot be done. Restricting ourselves to the one machine and its single language, there is no general solution to the halting problem.

It is interesting to note that the oracles in religion and mythology are usually imagined to have some special source of knowledge not available to the ones who come to question them. The oracle of myth and legend has some special inside knowledge that lets her solve the problems. Perhaps such oracles exist!

5.1 Commentary by Parabel and Cookie

Parabel. Well Cookie, do you buy this argument about oracles and lists and Losp and so on?

Cookie. Parabel, I am glad you asked. There is something really funny about that argument Kauffman gave in the second section of this text.

Parabel. What do you mean Cookie?

Cookie. Well look here. He had a list of good useful algorithms, and then he made a new algorithm **M** so that **M(N)** is the result of evaluating the **N**th algorithm at the number **N**. This is a rather nutty way to use all the known algorithms to do a computation. So Parabel, then he tries to put the algorithm on the list. Maybe $\mathbf{M} = \mathbf{A_N}$ for some **N** and this leads to a completely empty situation when Kauffman tries to evaluate **M(N)** under the assumption that $\mathbf{M} = \mathbf{A_N}$. Trying to compute $\mathbf{A_N(N)}$ makes the thing spin its wheels and we end up in an infinite loop!

Parabel. This is like trying to figure out who you are. I try to see who is Parabel and my mind says, why you are Parabel! That is not helpful!! Why should self-identity have anything to do with finding clever ways to study algorithms?

Cookie. It is rather astonishing how hollow and how significant is this ability to self-refer that we all take for granted. I could just sit here and spin:

"I am myself, tis I you see, I am I am I am I ...".

But curiously, I don't do that any more than two real players of hypergame would go on forever choosing hypergame. But when you make very strict rules for the algorithms to follow, then they behave like the automatic brooms in the sorcerer's apprentice and just keep multiplying nonsense.

Parabel. It is our freedom of speech that allows us to exit the infinite loop.

Cookie. Well Parabel, it seems that Kauffman is telling us that we had better be careful if we are only going to speak in one language. I am suspicious of this, after all I like to stay in English but I make up lots of new words and letters. Why just the other day, I made up the upside down A and the backwards E. And I make up the best new words all the time. Slithy, Outgrabe, Relicious, Wrench Monkey and so on. Everytime I do some new mathematics I make up at least ten new symbols and twenty new terms. My colleagues just love this! They immediately start chanting "Off with her head! Off with her head!" and I just know they are encouraging me to keep up the good work with my head and brain. So what is all this fuss about the limitations of using only one language?

Parabel. Kauffman first showed that algorithms are inexhaustible by this loopy argument. Then he said suppose you have to program the whole thing in one language. Then you get a contradiction. He showed that there was no way to decide on halting if we were restricted to use a single language. I think this is encouraging. We could restrict our algorithms to a single language and then use lots of fancy languages to analyze them!

Cookie. Well good luck to you there. By the way, I was thinking about another way to analyze the halting problem that goes straight back to Turing. You should look at [4]. He does it in a poem. Let me give you a flavor of it.

"Well, the truth is that P cannot possibly be,
because if you wrote it and gave it to me,
I could use it to set up a logical bind
that would shatter your reason and scramble your mind." [4]

Here P is that oracular algorithm that would decide the halting problem. Pullum has a poem that contains the proof. His proof is a rendition of the original proof of Alan Turing in 1936 [6]. Maybe we need a poem that goes beyond the proof!

Do you think you could make a super-machine that would decide about all the algorithms we can write in Losp. It would have to be very intelligent. It would have to decide about problems like the Collatz problem in Section 2. It would be a super-mathematician! I think we are all agreed that such algorithms are very unlikely to exist. But we will have a lot of difficulty proving that they do not exist if we give them complete freedom of language.

Parabel. But freedom of language is our right and so we have to admit that the real question for the decidablity of the halting problem or the question of mathematical incompleteness is still wide open. Can all well-stated mathematical problems be solved? It may be that they can be solved in systems of thought unknown, unseen and unmet by us as of yet. I am going to show you a poem that was found in Kauffman's papers from long ago.

Pattern by Louis H. Kauffman (Written 1962. First published in [2] page xi.)	
Pattern reigns supreme	*A symmetry with not a flaw*
To see and govern all the world	*It turns a graceful pirouette*
And regulate the flowing stream	*And cries again in places as of yet*
Of conscious thought	*Unheard, unseen, unthought or met.*
And perception's dream	*Its cadences repeat and ring*
It blends and fades and reappears	*To echo to a cosmic beat*
To form meaning from wondrous fears	*And find in this polyphony*
To captivate in human awe	*The guarded source of the discrete.*

Cookie. In chasing after the oracle that would solve all problems we are chasing after the principle of creation herself. All the ancient myths of oracles and myths of limitation as well apply here. It is good for us to join the quest and we must respect always the present but mutable limitations at each stage of the journey.

Parabel. It is that guarded source of the discrete that we have been analyzing in this paper. Kauffman is showing us the strange and simple structure that guards our knowledge and leads us onwards to new forms and new patterns in the endless round. I think we should think about the way our thoughts do swirl about themselves when we look for an answer.

Cookie. We seem to know when to stop in certain circumstances. In fact there is a notion of *imaginary value* due to George Spencer-Brown [1, Chapter 11] where this is understood to be a place in a circuit of distinction-operators where a value remains marked during a process of transition, only to turn off at the end of the process. The

mark that shuts itself off at the end of the process is regarded as an imaginary value. This mark acts to ensure that the circuit as a whole does not go into an infinite loop. The remarkable (and inevitable) property of this imaginary value is that it comes about through a part of the circuit that is seen to be an observer for the rest of the circuit's behaviour. But if you were to look at the structure of the circuit as a whole, you might be quite unable to tell that the circuit was observing itself in this way. You would just see a lot of operators in a circular interconnection.

Parabel. Are you suggesting that this advent of imaginary values in the behaviour of distinction operators is analogous to the way our brain and nervous systems are interconnected?

Cookie. Certainly I am suggesting this. We have evolved self-observing networks acting with imaginary values, that we call awareness, that can catch the system and shift it in directions away from infinite oscillations. It is of course only a speculation that this is the function of awareness. But I do believe that *form follows function*, and so we will find that everything that happens with us has its purpose.

Parabel. You could compare this use of imaginary values with the model for observation in quantum mechanics. There, an observation momentarily stops the dynamics of the Schrodinger equation's evolution. Then the Schrodinger evolution starts up immediately, but a macroscopic and repeatable mark has been registered for the observer to notice. The usual distinction between the micro and macro levels in a system can be understood differently from this point of view. The macro levels are those places where a value or values have become relatively fixed.

Cookie. These remarks of yours about quantum mechanics should be pursued. You are suggesting that there is nothing special about the "consciousness of the observer" that collapses the wavefunction, but rather that the system as a whole has built in self-observations that generate imaginary values as we have just discussed.

Parabel. We will have to formalize this in terms of a quantum version of networks of distinctions. This will give us something to do when we return to the void!

Cookie. It just occurred to me that if we could see into the future, then there would be no problem at all with the halting question!

Parabel. Oh no! Not that again. Do you really believe it?

Cookie. Sure. I would just look forward into the full infinite future and see if the algorithm would ever stop.

Parabel. Well then you would be a very special oracle, but you would not be a Losp program.

Cookie. Sure. But who cares about Losp programs, I just want the answers.

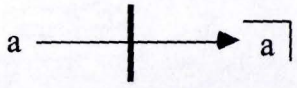
Parabel. Yes. I know you do, but there is something about this going on in our work. Take hypergame. It seems that we took hypergame to be a game *before* we found out that it was not a proper game. So we tried to make hypergame move into its own future.

Cookie. Now you're cooking! That is just what I mean. We use algorithms that anticipate a future. We do. We do. When I got married, I said "I do." and that determined my future by anticipating it in the present. When I make a contract at the bank for a loan it is my present action that creates the future existence of the loan. We anticipate our own actions and create our futures in the process. What we do is a

function of our future state. There is no fixed world out there where these algorithms either halt or do not halt. This is a fantasy!

Parabel. You should remember that you are nothing but a text on this page and stop getting such fancy ideas. But there is something right here. In order to forbid the set of all sets, you and I had to conceive of them. And having conceived of them, they exist, and so we did not really forbid them, we just made the Club Rules for dealing gently with them in the face of the logicians. Hypergame is a game whatever you say. But this business about the future and anticipating the future in terms of itself, I think we had better read those papers of Dubois and see how relevant his structures are to this discussion. [3]. The present need not be just a function of the past. It can be a function of both the future and the past. The present is a handshake or agreement of the future and the past.

Cookie. Lets get back to particulars, I would like to see how you think about circuits the have signal transmission and feedback. Then future and past are in a mutual interlock. We can represent such a system by using a version of [1] where the basic circuit element looks like this:



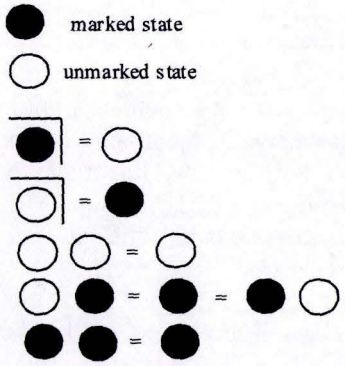
Here I am using the Spencer-Brown mark to indicate an inversion of value. The right-angle bracket is the mark, and when it surrounds the value a, it denotes the inversion of that value. The inversion of the marked state is the unmarked state. The inversion of the unmarked is the marked state. The dark vertical bar also denotes a mark as inverter. It is indicated in this fashion so that a mark (inverter) can be part of a larger network of interactions. The individual mark makes a distinction between its inside and its outside. Here that distinction is between the lines that go into the mark on the left and the lines that leave the mark on the right. Lines that enter the mark on the left can carry different signals, but all the lines that leave the mark on the right carry the same signal. Each individual mark in a network makes a distinction between input and output, and inverts the input to form the output.

Parabel. But wait a moment! I remember Spencer-Brown talking about this subject and he said that when he was making these designs there was a worry that the transistors in these inverters could act bi-directionally and so the arrow you describe could also go the other way!

Cookie. Well, you are right about that. And it is a subtle matter to design circuits that will work no matter what is the direction of the inversion. So I have decided for this discussion to impose the extra ordering condition so that we do not have to worry about this very important subtle point! I hope that is ok with you.

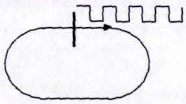
Parabel. I think you may be throwing out the baby with the bathwater when you do this, but ok.

Cookie. Allright then. Let us use a dark circle to denote the marked state and a white circle to denote the unmarked state.



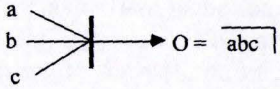
These equations indicate how the states interact.

Then we can have an always oscillating system like the following reentering mark.

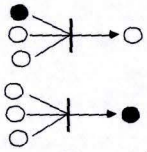


A marked signal transmits an unmarked signal as it goes through the inverter. This system will oscillate. It is the paradigm of a non-halting algorithm. It is literally an infinite loop.

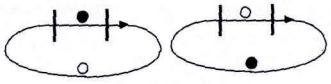
Parabel. More generally, we can consider a collection of inputs to the inverter, and a collection of outputs. *Balance* at the given inverter will mean that each output is the inversion of the juxtaposition of the inputs. This is illustrated below for three inputs a,b,c and one output.



The reentering mark can be construed as a circuit in which an inverter feeds back its output directly to the input. The result of such an interconnection is an oscillation between the initial state and its inversion. Such an interpretation assumes that there is a time delay between the production of the output and the processing of the input. If there is no time delay, then we are in a state of eternal change. The inputs are combine according to the rules for the marked and unmarked signals. Thus if a,b or c are marked then the output is unmarked. For example:

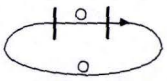


It is possible to have a circuit that does not oscillate. If we connect two of the inverting marks through one another then we get a circuit with a double-negation that has two balanced states as shown below.



This is form of a basic memory element. The balanced states are halted conditions for circuit.

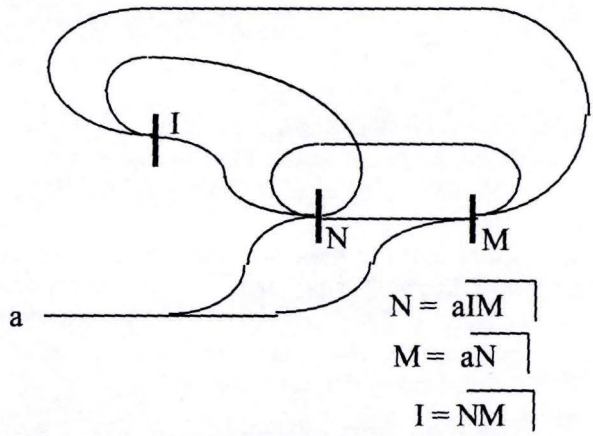
A more diabolical setting would be to have both sides simultaneously unmarked. The resulting state of the memory is then the unstable configuration shown below.



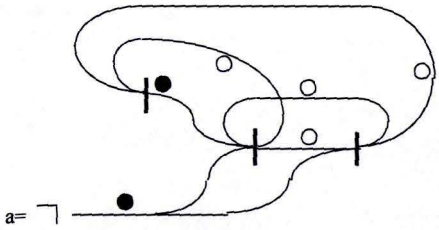
Each mark in the memory is unbalanced. The first mark to transmit a marked state will win the race and propel the memory into one of its two stable states. If it is possible for both marks to "fire" at once, we would arrive at the other unstable state where both sides are marked. In physical practice (with transistors) this simultaneity will not happen, and the above unstable state will fall to one or the other of the two stable states, just as it does in our transition model where one mark reacts faster than the other.

Parabel. It is this diabolical indeterminate state that I think is a precursor, or possibly a KEY to quantum computing. You see this state could be regarded as just oscillating very fast between both sides marked and both sides unmarked at some quantum level. And the slightest observation will flip it into one of the two balanced states for the memory. It is very much like a qubit in the realm of quantum computing. We'll have to discuss this in our next collision of texts. You do remember, Cookie, that we are only texts.

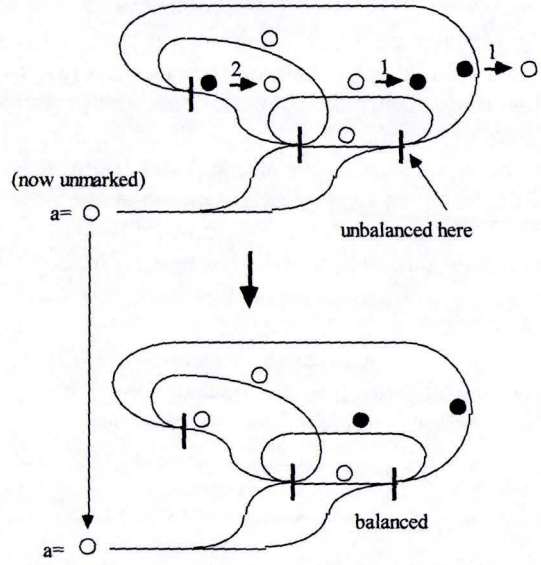
Cookie. As far as I am concerned, we are imaginary discussants and imaginary discussants *are* texts. Anyway, in practice, unstable memory conditions such as the above can occur, and it is interesting to see how to design a circuit that will transit to only one of the two possibilities. Consider the circuit below.



In this circuit we have eliminated the arrows that indicate direction of signals through the inverters and have used the convention that *signals travel through each inverter from left to right.* This suffices to fix all other directed lines. The memory consisting of **M** and **N** has an input a, and there is one more mark in the circuit labeled **I**. If a is marked, then **N** and **M** are unmarked, forcing **I** to be marked. This is a stable condition of the circuit so long as **a** is held in the marked state.



If now, we let **a** change to the unmarked state, then the circuit becomes unbalanced at **M** only, since **I** continues to put out a marked state.



In the diagram above, we show how the change of a to the unmarked state gives rise to a transition of **M** to the marked state (**1**) and that this forces a transition of **I** to the unmarked state (**2**). The resulting circuit is balanced and the transition to this state of the memory is determinate.

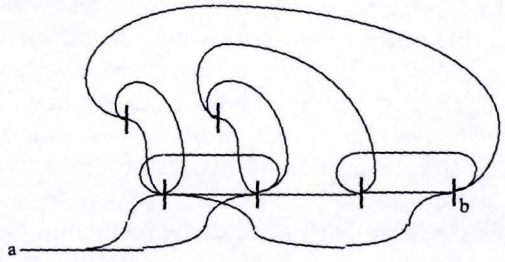
The addition of the mark **I** to the circuit enabled the determinate transition. In fact, **I** acts as an observer of **M** and **N** who feeds back the inversion of the or of **M** and **N** to the input to **N**. The result is that **I** is marked at the point of transition, holding the state at **N** in balance. For any choice of time delays, this condition of **I** can be arbitrarily small, but significant in forcing the transition.

We see that a circuit can be construed as a self-observing system, and that this condition of self-observation can radically influence its behaviour. The way the circuit behaves in the relation of I and N can be said to be an imaginary value. In terms of circuit design, we can use such imaginary values to influence the structure of the design and make otherwise indeterminate circuits determinate. We say that a pair of markers in a circuit has an imaginary value if there are transient states at one marker that influence the transition behaviour of the other marker. Just so the marker at N is held first by I and then after N changes to the marked state it is held by N and I is released to the

unmarked state. The sequence of these events is structural and independent of the time delays in any application.

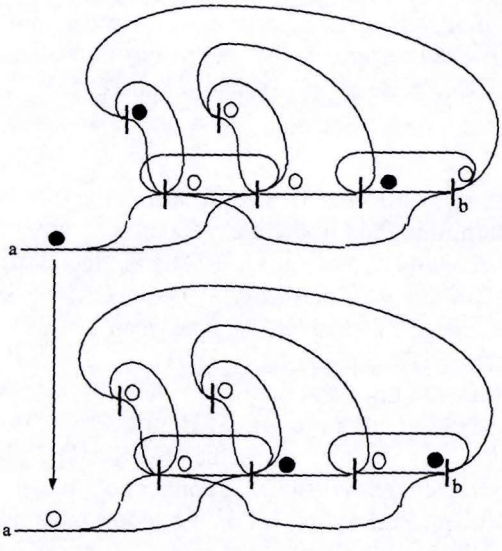
Self-observation can occur in a circuit without such transient states.
The circuit below is an example. It is a modulator in the sense of Spencer-Brown. That is a given frequency waveform input at a results in an output waveform of one-half the frequency at b.

Kauffman discovered this circuit in 1978 when studying Laws of Form. It is similar to circuits in Chapter 11 of Laws of Form, and it accomplishes the modulation without using any imaginary values using only six markers. Spencer-Brown gives an example in Chapter 11 of a modulator with six markers that uses imaginary values. The reader will enjoy making the comparison.



To see how this circuit operates, I have illustrated one stable state and one transition below. In showing the transition, I have only shown the end result. The reader will find that this is an example of a determinate transition.

Note also that in the next transition, the value of b will not change. There is a four-fold cyclic pattern in the transitions, with b changing every other time.



Modulators like this one are the building blocks for circuits that count and are often called flip-flops in the engineering literature. There is much more to say about this circuit structure and its relationships with computer design, information and cybernetics, but we shall stop here, only to note that this is an aspect of Laws of Form that goes far beyond traditional Boolean algebra, and is well-worth studying and working with as a research subject.

Parabel. To summarize this last part of our conversation, we have seen that modulators, built by circular interconnection of inverting operators (the process behind a distinction that is the distinction) can create counting circuits and hence universal Turing machines. I am sure that they can do much more, but it is very significant that they can do even this. They are a realm of reflexive recursive structure, just on the border of paradox that are contained away from paradox and non-halting conditions by self-observation and corresponding imaginary values in the circuit transition. All our arguments about the unsolvability of the halting problem in specific languages still apply to the Turing machines one might build from these modulators. But this construction gives one pause. The boundary is now shifted from sequential logic underlying the analysis of machines and languages to recursive, reflexive and self-referential structures that are in fact the primary support for all our attempts at the clarity of sequential and Boolean thought. Does the halting problem remain unsolvable at this more fundamental level?

Cookie. You have me spinning now.

Parabel. That is good. Maybe you can enter a quantum state. It is time to say goodbye.

Cookie. Goodbye.

Acknowledgement

This paper is a modified version of a paper by the same author for the Festschrift for Soren Brier. The main changes are in the extension of the last section to include a discussion of imaginary Boolean values in relation to halting.

References

1. G. Spencer-Brown, Laws of Form, Alan and Uwin Pub. (1969).
2. L. Dennis, The Pattern, Integral Publishing (1997).
3. D. M. Dubois, Incursive and hyperincursive systems, fractal machine and anticipatory logic. Computing Anticipatory Systems. CASYS 2000 – Fourth International Conference. Published by The American Institute of Physics, AIP Conference Proceedings 573, pp. 437- 451 (2001).
4. G. K. Pullum, Scooping the Loop Snooper, <http://www.lel.ed.ac.uk/~gpullum/loopsnoop.html>
5. The Sorcerer's Apprentice,<http://en.wikipedia.org/wiki/The_Sorcerer's_Apprentice>
6. A. Turing, On computable numbers, with an application to the Entschneidgungsproblem, Proceedings of the London Mathematical Society, Series 2, 42 (1936), pp. 230-265.