Lecture Notes Series

Robert Geroch

Unsolvable Problems

1990 Lecture Notes

MINKOWSKI
Institute Press

"Mathematics can never become routine; by its very nature there will always be required genuinely new and more sophisticated insights." This sounds like a good topic for an afternoon's debate. But, believe it or not, there is actually a *theorem* in mathematics to this effect! This result, called Godel's theorem, is one of the greatest discoveries in mathematics of the twentieth century. And - perhaps even more remarkable - the proof of this theorem can be understood by anyone, without any knowledge of mathematics at all. This proof is the subject of this book.

ISBN 978-1-927763-02-5

http://minkowskiinstitute.org/mip/

Robert Geroch

Unsolvable Problems

1990 Lecture Notes

MINKOWSKI
Institute Press

Robert Geroch
Enrico Fermi Institute
University of Chicago

Cover: Lecture notes are often written in similar environments

ISBN: 978-1-927763-02-5 (softcover)
ISBN: 978-1-927763-05-6 (ebook)

Minkowski Institute Press
Montreal, Quebec, Canada
http://minkowskiinstitute.org/mip/

For information on all Minkowski Institute Press publications visit our website at http://minkowskiinstitute.org/mip/books/

Contents

1. Introduction

Here is a little game.[1] Begin with string consisting solely of letters “a” and “b”,e.g., “ababbbabaa”. The leftmost letter of this string is “a”. So, append the letters “aa” to the right of the string and delete the leftmost three letters. There results “bbbabaaaa”. The leftmost letter of this new string is “b”. So, append the letters “bbab” to the right and delete the leftmost three letters. There results “abaaaabbab”. Now continue in this way, at each step appending “aa” or “bbab” on the right according as the leftmost letter is “a” or “b”, and deleting the leftmost three letters.Thus, in this example the next few steps will yield “aaabbabaa”, “bbabaaaa”, “baaaabbab”, and “aabbabbbab”.

What would happen were we continue to play in this way? Clearly, there are just three possibilities. First, we could ultimately arrive at the empty string – the string with no letters – thus terminating the game. This would happen, for example, were the initial string just “aa”, for the next step would yield just “a”, and the next step the empty string. Second, the string could ultimately repeat itself. Then we would continue to play in a cycle, going through the same sequence of string over and over. This would happen, for example, were the initial string “aabbab”, for the next step would yield “babaa”, and the next step “aabbab”, the same as the initial string. Finally, it is possible that the game go on forever, without terminating or cycling.

More generally, one specifies a tag game by giving the alphabet for the game (in this example, “a” and “b”; but it could consist of more letters), the rules for what is to be added on the right according to which alphabet letter is leftmost (in this example, “aa” if “a” is

[1] From Ann Yasuhara’s book on recursive functions.

leftmost and “bbab” if “b” is leftmost; but there could be given any string), and the number of leftmost letters to be deleted (in this example, three; but it could be any number). Then given the tag game and the initial string, one of the outcomes above must occur. How would one determine which outcome will occur in a given case? Well, one could merely play the game. If it terminates or goes into a cycle, one would notice it, and this would be the answer. But what if the game continues, for days, months, decades? Clearly, mere play of the game will not always answer the question of which outcome will occur. Alternatively, one might devise some general argument – not involving actual play – for what the outcome will be. (A simple example is the tag game with alphabet “a” and “b”, with “a” → “aaaa” and “b” → “bbbb”, and with three letters deleted at each step. Then every game, no matter what the initial string, goes on without terminating or cycling.)

For any given tag game and initial string one presumably could, given sufficient time and effort, devise some way to determine which of the three outcomes will occur. Let us now ask a more sophisticated question. Can we do more than this? Can we devise some general procedure that will determine, for *any* given tag game and initial string, which outcome will occur?

This last question goes to the heart of the subject of this course. We are interested in well-posed questions (such as, in the example above, the question of whether a given tag game on a given initial string terminates, cycles, or neither), and whether there are procedures for answering those questions (as is asked above). The remarkable result – that has extensive implications both within and without mathematics – is that one can in fact specify well-posed problems that can be solved by no procedure whatever. (In fact, there is no procedure to answer the tag-question above, but the actual example we shell give will be slightly different.)

It is our purpose to understand this result. The first step is to gain a somewhat firmer sense of what “well-posed problem” and “procedure” are to mean.

2. Games

Let us consider games of the following type. There are two players – call them “x” and “o”. There is a board, visible to both players at all times, the possible configurations of which give the entire status of the game at any one moment. The players alternate turns, and each player in his turn makes a move – makes one of certain allowed changes in the board configuration. Certain board configurations are designated as “x wins”, certain as “o wins”, and certain as “draw”. If, as the game proceeds, one of these configurations is reached, then the game terminates, with the corresponding outcome. If none is ever reached, then the game is declared a draw.

Examples of games that fall within the framework above are tic-tac-toe, checkers, chess, nim (see problem 2 Set 1), and five-in-a-row (Set 1). An example that doesn’t is poker (concealed hands violate the board-configuration provision).

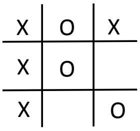
Such games provide good examples of well-posed problems, their solutions, and the issue of whether there are procedures for finding the solutions. The problem is posed to a given player in his turn by the board configuration: He must decide which of the allowed moves to make. The solution to the problem is to make the best move – one that will result in a win if possible, but if not that a draw. More on this in a moment. Whether there is a procedure for solving the problem asks whether one can give explicit instructions such that a technician, merely following those instructions and having no insight, can play expertly, i.e., in every case make the best move (or one of the best, if there are several such).

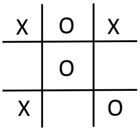
Let us now detour for a moment to describe in more detail the “best move”. Consider the following rules:

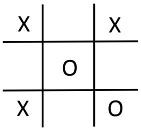
1. A board configuration is x-winnable if it is one of the “x-wins” configurations above.

2. A board configuration is x-winnable if it is x’s turn, and there is *some* move for x that results in an x-winnable configuration.

3. A board configuration is x-winnable if it is o’s turn, and *every possible* move for o results in an x-winnable configuration. The x-winnable configurations, roughly speaking, are those for which “x, if he plays carefully and skillfully, is assured of winning no matter what o does”. In tic-tac-toe, for example, the first configuration below is x-winnable, by rule 1.

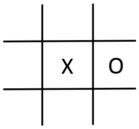






Therefore, the second configuration with x’s turn, is x-winnable, by rule 2. Finally, the third configuration, with o’s turn, is also x-winnable, by rule. (No matter what o does in his turn, there will result a configuration, like that of the second figure above, that is x-winnable.) In a similar way, we define the *o-winnable* configurations. Thus, for example, a configuration is o-winnable if it is an “o wins” configuration; or, if it is o’s turn and there is some move for o that results in an “o wins” configuration; or, if it is x’s turn and, no matter what move x does, there is some move for o that results in an “o wins” configuration; or, if it is o’s turn and there is some move for o such that, no matter what move x then does in his turn, there is some move for o that results in an “o wins” configuration. It continues in this way. In short, an o-winnable configuration is one from which o can, against any play by x (good play, bad, or whatever), force eventually an “o wins” configuration.

Finally, a configuration that is neither x-winnable nor o-winnable will be called *drawable.* The drawable configurations, roughly speaking, are those from which neither player, with sufficiently skillful play, will lose. In tic-tac-toe, for example, the initial board, before anyone has moved, is drawable. If x takes his first move at the center, then the configuration is still drawable. If, however, o then



moves to the side, then the configuration becomes x-winnable. (A move by o to the corner would have made another drawable configuration.)

Suppose that it is x's turn. Let, first, the board configuration be x-winnable before x's move. Then, by rule 2 above, there must exist some move for x such that the configuration is x-winnable after x's move. This would be a good move for x. There may, of course, also be moves for x that result in a drawable or even an o-winnable configuration. These are clearly bad moves. Let, next, the board configuration be drawable before x's move. Then there is no move for x that results in an x-winnable configuration (for by rule 2, were there such a move then the configuration would already by x-winnable, not drawble); and there must exist some move for x that results in a drawable configuration (for were there none then the configuration would already be o-winnable, not drawable). Thus, a good move for x (the best x can do) is one that results in a drawable configuration after x's move. There, may, of course, also be moves for x that result in an o-winnable configuration. These are clearly bad moves. Finally, if the board configuration before x's move is o-winnable, then it doesn't make any difference what x does: The configuration after x's move will in any case again be o-winnable.

We can summarize all this by saying that one can never, on one's move, improve the situation (go from winnable by the opponent to drawable, or from drawable to winnable by oneself). One can always keep it the same. There may also be moves that keeps the situation worse. By a *good* move, we mean one that keeps the situation the same, i.e., that does the best one can do. (You are invited to think through these definitions – of x-winnable, o-winnable, drawable, and a good move – in the case of tic-tac-toe. You should be able to convince yourself that, at least for this game, these terms all take on their usual meanings.)

So, at least for games that fall within our framework, there always exists at one's turn a good move – one that will not cause one's situation (winnable for oneself, drawable, winnable for opponent) to deteriorate. The problem is to find such a good move. For any one specific board situation one could, with some luck and hard work, perhaps come up with a good move. But we now ask for more: We ask for a procedure by which, given the board configuration, one can generate a good move. Is there, given the game, such a procedure?

In the case of tic-tac-toe, there certainly does exist a procedure for finding good moves. Indeed, we all, at least intuitively, know such a procedure. It is not even that hard to write out a procedure in detail. (See problem 1, Set 1.)

Does there exist a procedure for finding good moves in checkers? It is not as obvious as tic-tac-toe, for we are not as skillful checker players. But, I claim, there must exist such a procedure. The reason is that there are just a finite number of possible board configurations in checkers. (This is clear, for each square of a checker board can be in just one of five possible situations – blank, my single piece, my king, your single piece, or your king. Hence, the number of possible configurations of a checker board cannot exceed 5 x 5 x ... x 5 (32 times), which is about 23,300,000,000,000,000,000,000.) Having noticed this, one can, at least in principle, use the following procedure. Purchase a book with 23,300,000,000,000,000,000,000 pages. On each page, draw the corresponding configuration of the checker board (so every configuration is included). Then, at the bottom of each page, write out a good move for that configuration. This is a procedure in every sense of the word. A technician can well play expert checkers using our book. (At each turn, he merely looks up the board configuration in his book, and makes the corresponding move.) Note that a similar argument could have been used for tic-tac-toe (but was unnecessary there, for we actually knew the strategy).

Thus, we could conclude so far that for every game within our framework there exists an expert strategy – a choice of a good move for every board configuration. (This is essentially by definition of "good move".) Further, in the case of a finite number of board configurations, there always exists a procedure for finding the good moves. A suitable procedure is merely to list every possible board configuration, and for each one, a good move for that configuration. In particular, then, there exists a procedure for expert play in chess.

But what of games with an infinite number of possible board configurations? One such game is nim. (See problem 2, Set 1.) There is a procedure for expert play in this game, as we saw in that problem. In fact, on x's turn, the x-winnable configurations are those for which the remainder on dividing the number of beans in the bowl by three is one or two; and the o-winnable configurations are those for which the remainder is zero. There are no drawable configurations (which figures, since there are no draws in this game).

As a final game, consider five-in-a-row (problem 4, Set 1). This game also has an infinite number of possible board configurations (for the size of the grid on which the marks are made is potentially infinite). Because of this feature, it would be hopeless to try to give a procedure for expert play by writing a “book”, because there is no limit to the number of pages that would be required. Does there exist a procedure for expert play in this game? I do not know the answer. The game is very much more complicated than nim, so the rules for expert play would certainly be substantially longer than the rules for nim.

We conclude that, for games within our framework having an infinite number of board configurations, there is no way to assert (as with checkers) that there does exist a procedure for expert play. We may be able to find such a procedure and show it works (as is the case with nim), or we may simply fail to find a procedure, and then not know if there is one at all. So far, at least, we do not have the tools to show in any case that there exists no procedure.

3. Arithmetic

We now turn to a new class of problems – those involving arithmetic. These problems will on the one hand serve to refine our understanding of the notion of a "procedure". Indeed, problems involving arithmetic are better suited than those involving games to illustrate what we wish to mean by a procedure. Further, some of the methods we use here will arise later in our formal discussion of procedures.

By an integer we mean a (positive, negative, or zero) whole number. Thus, 13, –845621, 0 and 2 are integers, while 2.3, 1/2, and π are not. The basic idea is now quite simple: We wish to introduce various questions and problems about integers, and then consider what the solutions are and whether or not there exist procedures for obtaining those solutions.

Is the integer 6 even? We certainly know what the answer is! Is there a procedure to produce the answer? (Recall that a procedure is to be instructions that can be followed mechanically, with no insight, to produce the desired answer.) There certainly is such a procedure. Here is one: "Just say 'yes'. " This is indeed a set of instructions, and it does indeed yield, mechanically, the desired answer. What about the instructions: "Place 6 beans in a pile, and remove them two at a time. If at the end there remains one, say 'no'; if none, then 'yes'."? This is also a perfectly good procedure. But what makes it good is not that it is germane to the problem, but only that it gives the right answer. Thus, the following is also an acceptable procedure for answering our question: "Place 17 beans in a pile, and remove them five at a time. If, when no more can be removed in this way there are still beans in the pile, then say 'yes'." In short, for a procedure we are only interested in whether we get the right answer or not.

We are not interested in whether the technician is doing something germane, interesting or whatever. It is only the final answer emitted that counts. The situation here is somewhat analogous to that in tic-tac-toe, in which one can give a procedure for expert play by merely listing every possible board configuration and, for each configuration, a good move. This is an acceptable procedure. It is not necessary, for a good procedure, that the technical actually do anything germane to tic-tac-toe (such as make arguments about "if I do this, then my opponent will do this..."). In short, returning to our original question, a suitable procedure for answering it is any explicit instructions for a technician that result in his producing eventually the answer "yes".

Is the number 7 even? All the remarks above apply, and we have only to produce instructions that result in "no".

Given integer between 1 and 10, is it even? It will be recognized that this is not a single question, but rather is ten questions ("Is 1 even?", "Is 2 even?",..., "Is 10 even?") all assembled together. So, there is not a single answer, but ten answers, one of each choice of integer. There clearly is a procedure for answering any one of these questions (and, indeed, we have just done two of them, above). But now we ask for more. We ask for a single procedure, given once and for all to our technician, such that our technician can, armed with that procedure, answer all ten questions correctly. Let's try a few possibilities. Consider "Just say 'yes'." This is certainly a procedure, but it is not a correct procedure for our problem (for, e.g., it gives the wrong answer if "7" is presented). Our procedures, then, had better require that the technician actually use the number presented. Here is a suitable procedure: "If the given integer is 2, 4, 6, 8, or 10, then say 'yes'; if 1, 3, 5, 7, or 9, then say 'no'." Note that this does in fact produce the correct answer – and that is all we care about in the procedure. Another suitable procedure, for example, is "Place in a bowl a number of beans equal to the given integer, then remove them two at a time until this is no longer possible. If a bean then remains, say 'no'; if none remain, say 'yes'." But, again, the procedure need not be germane to the problem. Thus, here is a suitable procedure: "If the number, when spelled out, either ends in an 'e', or contains two 'e"s, then say 'no'; otherwise, say 'yes'." Again, the situation here is similar to that of tic-tac-toe: Any procedure that results in expert play is ok, whether or not that procedure gives any particular insight into the game itself. In particular a mere list of board configurations and corresponding good moves is acceptable.

Given any positive integer, is it even? This will be recognized as not one question, not ten questions, but rather an infinite number of questions. Our previous methods for obtaining procedures do not work in this case. "Just say 'yes'." does not, of course, always produce the correct answer. And a listing of all possible integers that might be presented, together with the answer to give for that integer, is not possible with an infinitive number of possible integers. Is there, then, a procedure for answering these questions? There certainly is. One procedure is bean-counting: "Place in a bowl a number of beans equal to the integer given, and remove them two at a time until that is no longer possible. If a bean then remains, say 'no'; if not, say 'yes'." In fact, there are easier procedures. For example: "If the given integer ends in 2, 4, 6, 8, or 0, then say 'yes'; if in 1, 3, 5, 7, or 9, then say 'no'." Note that this last procedure "just happens" to work. Being even is quite different from ending in 2, 4, 6, 8, or 0. But it just happens to be the case that the two are equivalent. This example just further illustrates our point: All that matters of a procedure is that it produce the right answer, not whether the details of that procedure are relevant to the problem, or interesting, or anything else. The situation for this question is rather like that of the game of nim. In the game, there are an infinite number of board configurations, in analogy with an infinite number of questions here. Yet, despite the infinite numbers in both cases, we can find a procedure – for expert play in nim, and for answering the questions here.

Given a positive integer, is it evenly divisible by 3? This question is rather like that above, except that we need for it a somewhat a different procedure. Divisibility by 3 is not revealed by the final digit alone. We could, for example, use bean-counting. Or, one could write out a procedure that includes an explanation of how to do long division (for that, after all, is itself a procedure). Then, our procedure would dictate that the given number be divided by 3, and the remainder examined. Of course, our instructions for long division would have to include the multiplication table, but that is ok: We merely include it.

For the next few examples, we need the following definition. An integer two or greater is called *prime* if it is evenly divisible only by the integers one and itself. Thus, for example, 2, 3, 7, 19, 97, and 5737 are prime integers, while 4, 6, 35, 111, and 5767 are not.

Given a positive integer, is it prime? This is, of course, an infinite number of questions. There is, as it turns out, a procedure for answering them. Suitable instructions might direct that the technician divide the given integer in turn by each integer from 2 up to one less than the given integer, and find in each case the remainder if any. If in every case there is a remainder, then answer "yes"; if in any case there is none, then answer "no". Thus, for example, were the given integer 13, then one would try to divide it, in turn, by 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12. In each case one would obtain a remainder, and so one would answer "yes". Of course, these instructions for our technician would also have to include detailed instructions for how to perform long division (including, e.g., the multiplication table), but there is no difficulty in including this. Alternatively (or actually, if you think about it, the same thing), we could direct that the technician write down in order the integers from 1 up to the given integer. Cross out every second in this list, then every third, every fourth, and so on up to one less than the given integer. If the given integer is not in this process crossed off, then say "yes"; if it is, say "no". So, for example, if the given integer were 13, the technician would write 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13. He would then cross off every second number (namely 2, 4, 6, 8, 10, 12), then every third (namely 3, 6, 9, 12), and so on, up to every twelfth. In this process, "13" would never be crossed off, and so the technician would answer "yes".

Given a positive integer, is it the sum of any two prime integers? There is a procedure for answering this question. Suitable instructions might direct that the technician consider in turn each integer from one up to one less than the given integer, and determine whether that integer, and the given integer minus that one, are both prime. If in any case they are both prime, answer "yes": if never both prime, answer "no". Of course, the technician could determine whether these various integers are prime, using the procedure we just described above. Thus, for example, if the original integer were 16, then the technician would ask "Are 1 and 15 both prime?", "Are 2 and 14 both prime?", 'Are 3 and 13 both prime?", and so on. In this example, one would find a case in which both are prime (namely, 3 and 13), and so the technician would answer "yes". On the other hand, were the given integer 11, the technician would, as one can check directly, ultimately answer "no".

Given a positive integer, is it the difference of any two prime integers? This question is of course quite similar to the one above, and so one might try a similar procedure. Thus, one might direct that the technician consider in turn each integer beginning with one more than the given integer, on up, and determine whether that integer, and that integer minus the given integer, are both prime. If in any case they are both prime, then answer "yes"; if never both prime, answer "no". Thus, for example, if the original integer were 16, then the technician would ask "Are 17 and 1 both prime?", "Are 18 and 2 both prime?", "Are 19 and 3 both prime?", and so on. In this example, one would find a case in which both are prime (namely 19 and 3), and so the technician would answer "yes". Yet this is not a procedure! The problem is that the technician is being asked to test an infinitive number of cases (from one more than the given integer "on up"). It all works very well if, as in the example above, a case is found with both prime, for then the technician can stop and say "yes". But what if no such case is found – after a day, after a month, after decades? When will the technician be directed to say "no"? Indeed, if you have a lot of time on your hands, you may wish to play technician, say, starting with the given number 19. In short, without a guarantee that the technician's efforts will at some point come to an end, this is not a procedure as we wish to use this term. The key difference between this example and the preceding one, of course, is that before there were only a finite number of cases to be tested for any given integer, whereas here there could be an infinite number. So, our given candidate for a procedure fails. That does not mean that there exists no procedure, only that we have not found one yet. I do not know whether or not there exists such a procedure.

Given a positive integer, is it the largest prime? This question appears on its face to be very similar to the previous question. Thus, the obvious "procedure" here would be to test for primeness each positive integer starting with one more than the given integer, on up. If and when one is found to be prime, answer "no". But this, as it stands, may not be a procedure, for we have as yet no guarantee that the technician's task will not go on forever, But what sets this question apart from the previous one is the following:

Theorem. There is no largest prime integer.

Proof: Let n be any positive integer. We show that there is a prime integer larger than n. Consider the number z = 1 x 2 x 3 x . . . x n + 1. That is, z is the result of multiplying 1 times 2 times 3, and so on all the way up to the given integer n, and then adding 1 to the result. Now z is either a prime integer or it is not. If it is prime, then – since z is certainly larger than n – we have indeed found our prime integer larger than n. But what if z is not prime? Then by definition, we may write z as a product of two integers. If either of these is not prime, we may in turn write it is a product of two other integers. Substituting, we will have an expression for z as a product of three integers. Continuing in this way, always replacing any non-primes in our expression by products of other integers, we will eventually end up with an expression for z as a product of prime integers. What prime integers can appear in this final expression? Well, 2 cannot appear, for z is not evenly divisible by 2. (Indeed, z was originally given as a product of 2 and something (namely 3 x 4 x . . . x n) plus 1. So, the reminder on dividing z by 2 is 1.) Similarly, 3 cannot appear in this final expression; nor can 4; and so on up to n. That is, the prime integers that appear in our final expression must all be greater than n. Thus, we again find a prime integer greater than n.

What we have shown, then, is that given any positive integer n – no matter how large – there exists a prime integer larger than n. Clearly, then, there can be no largest prime integer.

Armed with this result, let us now return to our original question. We can claim, now, that what we originally gave for this question was indeed a procedure. Our theorem guarantees that the efforts of the technician will indeed terminate, and so he will indeed eventually produce an answer (in fact, “no”). But there are, of course, alternative procedures for this question. We could, for example, tell the technician “Just say ’no’.” What we learn from this example, then, is that a set of instructions, to be a procedure, need only have the property that it does indeed terminate and lead to the correct answer. It need not be obvious to the technician that the instructions will terminate.

Given positive integer n, do there exist positive integers x, y, and z, such that $x^n + y^n = z^n$? (That is x raised to the nth power, plus y raised to the nth power, must equal z raised to the nth power.) If, for example, n = 2, then the answer is “yes”. Indeed, set x = 3, y =4, and z = 5. We have 3^2 = 9, 4^2 = 16, and 5^2 = 25, while 9 + 16 = 25. For n = 3, the answer, it turns out, is “no” (but the proof is somewhat difficult). For general n, the answer is not known. Is there a procedure to answer this question. The obvious method would be, given n, to try various x's, y's, and z's. But this does not look promising, for we have no way to guarantee that the process terminates. Thus, it seems difficult offhand to come up with any suitable procedure. But note that, on the other hand, we are not in a position to assert that no procedure exists. There is a fair chance, for example, that the only n's that work are n = 1 and n = 2, all others failing. Then, suitable instructions would be "If n is either 1 or 2, say 'yes'; otherwise, say 'no'." Here, then, is an example of a question such that we just do not know whether or not there exists a procedure for answering it. Some day, perhaps our knowledge of mathematics will have risen to the point that we do know. We emphasize, however, that, for a given question, there either is a procedure or there is not. It is not a function of our state of knowledge in mathematics. As we learn more mathematics, we will merely know in more cases whether or not there exists a procedure.

It is true that, for all n greater than 2, there are no positive integers, x, y, and z, such that $x^n + y^n = z^n$? It is not presently known that the answer to this question is. Does there exist a procedure for answering it? Note that this is a single question: The answer is either "yes" or "no". Thus, there certainly does exist such a procedure. It is either the one with instructions "Say 'yes'.", or the one with instructions "Say 'no'." Of course, we do not happen to know, currently, which of these is the correct procedure. Perhaps we will some day. But in any case we do know that one of them is, and so we know that there *exists* a procedure to answer this question. In short, we wish to distinguish between the existence of a procedure for answering a question and the act of actually answering the question. We wish to allow the existence of a procedure if we can show it exists, whether or not we happen to have it in hand.

Given positive integer n, what is the nth digit in the decimal expansion of π (= 3.1415926535...)? (Thus, for example where n given as 7, then the correct answer is "6", the seventh digit in the above expansion.) There does in fact exist a procedure for answering this question. There are available any number of formulas that allow one to compute the value of π to any desired degree of accuracy.

Given integer from 0 to 9, does it occur ten times in a row in the decimal expansion of π? We do not happen to know the answer to this question for any value of the integer. Nonetheless, there is a procedure to answer the question. We can basically consider all possibilities. Thus, we have “Say ‘yes’.”, “Say ‘yes’ for 0, 3, 5, and 6; and ‘no’ for 1, 2, 4, 7, 8, and 9.”, and so on. There is a grand total of 1024 such possibilities. Since each of these ten questions has an answer (either ‘yes’ or ‘no’), one of these possibilities is the correct procedure. Hence, there *exists* a procedure to answer the question.

Given any positive integer, does it recur ten times in a row in the decimal expansion of π? (Thus, for 37, we ask whether “37373737373737373737” appears anywhere in the expansion.) The above procedure will not work here, for there are an infinite number of questions, and so we cannot make a list of answers. I do not know whether there is a procedure for answering this question. It may, for example, turn out that the answer is always ‘no’, or always ‘yes’, in which case there certainly does exist a procedure. On the other hand, it is at least conceivable that the yes’s and no’s are so scattered around in the answers that no procedure will reproduce them.

4. Encoding

There is clearly an enormous variety of questions with respect to which one could ask for procedures. A first step in understanding procedures better would be to standardize these questions. (Ultimately, indeed, we shall even standardize the procedures themselves.) For example, some of our earlier questions required as input a board configuration; those more recent, a positive integer. A question such as "Given positive integers a, b, and c, does a + b = c?" requires as input several integers. It would be helpful, then, if we could have a standard input – as opposed to all this variety – for our questions. This standardization is carried out by encoding various types of information into integers.

We seek first a method for encoding two positive integers into a third. That is, we wish a rule that allows us, given the two integers, to compute a third – but this rule must be such that, given only the third integer, one can "decode" to determine the original two. (Thus, for example, adding the two integers would not do, for one could not in general recover the two addends given only their sum.) We shall adopt the following rule: Write the two integers one after the other, but between them insert a number of "0" 's equal to the total number of digits of the two integers combined. Thus, for example, were the integers 30060 and 72, then we would encode these to obtain the single integer 30060000000072. (Note that we insert 7 "0" 's between the two integers, since those integers together have 7 (= 5 + 2) digits.) This rule indeed satisfies our criterion. To decode, take the total number of digits of the given number, divide by two, find the first that many "0" 's in a row, starting from the right, and cross them out. There results the original two integers, Thus, for example, given 1900000300, there are ten digits. Removing the first five o's in a row, starting from the right, we are left with ‘19 300”. These are the original two integers. Of course, there are many integers that cannot be obtained as a result of encoding others, e.g., 2064 and 2000064.

Given positive integer n, is it the result of encoding positive integers a and b, with a greater than b? There is, of course, a procedure to answer this question. But note the role that encoding plays. This is really a question about two integers (a and b), but, by use of encoding, it has as its input a single positive integer, n.

How would one encode three positive integers into a single integer? There is a simple way. First encode the second and third, to obtain a new positive integer. Then encode the first and that integer. Thus, for example, let the three numbers be 111, 60, and 21. We first encode the second and third, to obtain 60000021. We then encode 111 and 60000021, to obtain 1110000000000060000021. From this single integer we can, in turn, recover our original three. Indeed, we first decode 1110000000000060000021, to obtain 111 and 60000021. Then we decode the latter, to obtain 60 and 21. Thus, we recover our three integers, 111, 60, and 21. Note that there will never be any ambiguity in this, as long as one knows the code.

Similarly, to encode four positive integers, we first encode the third and fourth, then the second with the result, then the first with the result. There results a positive integer that can be decoded to obtain the original four. It works similarly with any given number of positive integers.

Given positive integer n, is it the result of encoding three positive integers, a, b, and c, with a + b = c? There is of course a procedure. By encoding, this question (actually about three integers) is made to require as input just a single integer.

In each of the examples above, we are given, beforehand, the number of integers that are being encoded. Thus, for 1110000000000060000021, if we knew it was to be decoded into three integers we would obtain 111, 60, and 21; while if we knew it was into two integers we would obtain 111 and 60000021. Is there some way we can arrange that not only the integers themselves, but also the number of integers, is included in the encoding process? That is, we wish to be given a single integer – and no other information – and obtain from it both how many other integers are to be obtained, and what they are. This is actually quite easy. Let there be given any list of positive integers we wish to encode. Count the number of integers in this list, append this number to the beginning of the list (to obtain a new list, with one more integer), and finally encode this new list as above. Thus, for example, let the original list of integers to be encoded consist of 13, 9, and 6. There are three integers in this list, and so our new list consists of 3, 13, 9, and 6. Finally, the result of encoding these four integers is 30000000000000130000009006. It is clear that all this can be decoded. Thus, in the example above, given 30000000000000130000009006, we first decode once, to obtain 3 and 130000009006. The first integer, “3”, tells us how many integers remain encoded in the second integer, 130000009006. Thus, we decode this twice more, to obtain finally ‘13”, ‘9”, and “6” our original three integers. Try with 200000000010300005.

Given positive integer n, is it the encoding of a list of two or more integers such that the last is the sum of the others? Here is a question about an indeterminate number of integers that, through encoding, is rendered as a question about a single integer.

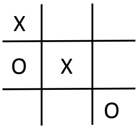
Much more detailed information could also be encoded into a single positive integer. Consider, for example, the novel *War and Peace.* One could first assign numbers to the various symbols that appear in the book, e.g., “a” = 1, “b” = 2,. . ., “z” =26, “.” = 27, “?” = 28, “ ” = 29, and so on. Since the book itself is merely a sequence of such symbols, we could replace it, first, by the corresponding sequence of integers. Thus, if the book begins “It was a . . .”, our sequence would begin 9, 20, 29, 23, 1, 19, 29, 1, . . . Now, we simply take the resulting sequence of integers, and encode it as above (using the final version, for an indeterminate number of integers, since we do not know, a priori, how many there are to be) into a single integer. *War and Peace*, in this way, is rendered as a single positive integer.

One might also encode information about the various games discussed earlier. Consider, for example, tic-tac-toe. Let us number the squares of the board as shown on the right. Then we may represent a given board configuration by a nine-digit number, where each digit represents the corresponding square. Each digit, in turn, will tell us what is in that square, say using the following code: “x” = 1, “ ” = 2, “o” = 3. Thus, for example, the board configuration shown on the right would be encoded as the integer 122312223. Clearly, one

1	2	3
4	5	6
7	8	9

can decode to determine the board configuration from the integer. One could also include whose move it is, e.g., by including a tenth digit on the right that is "1" if it is "x" 's move, and "3" if "o" 's. Thus, the above configuration, with "x" 's turn, would be represented 1223122231.

Given positive n, is it the encoding, above, of a tic-tac-toe board configuration and whose move it is such that this configuration is x-winnable? Drawable? These questions about tic-tac-toe thus, by encoding, become questions about positive integers.

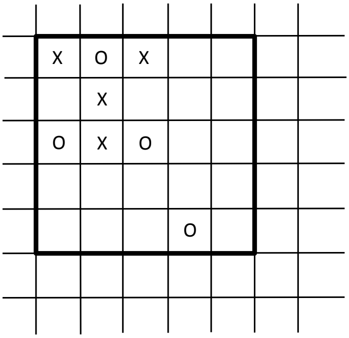


We could represent a "move" in tic-tac-toe by giving a 19-digit number consisting of the initial board configuration (first 9 digits), whose turn it is (next digit) and the final board configuration (last 9 digits). Thus, if it is x's turn above and x takes the upper right hand corner, this move would be represented as 1223122231121312223. Clearly, one can decode to recover the move in the more familiar form.

Given positive integer n, is it the encoding, above, of a good move in tic-tac-toe?

In a similar way, one could encode positions and moves in checkers. (Number the squares on the board, assign numbers to represent what can be on those squares, and then represent the board configuration by some large integer, etc.) One could encode a nim move by first determining the number of beans in the bowl before the move, whose turn it is (say, "x" = 1, "o" = 3), and the number of beans in the bowl after the move. These three integers could then be combined, using our encoding above, into one. Thus, for example, 28000000300027 is a good move in nim, while 28000000300026 is not.

Finally, let us consider five-in-a-row. The complication here is that, as compared with checkers, say, the board is infinite. We may represent a board configuration in five-in-a-row as an integer, for example, as follows. Given a board configuration, first draw on the board a large square that includes all moves made so far. Then number the small squares within that large square from left to right, top to bottom, as in tic-tac-toe. To encode the board configuration, first give the integer that represents the length of a side of the square, and then the integers that represent what ("x" = 1, " " = 2, "o" = 3) is on each small square. Then take all these integers, and encode them into a single integer as above.



Consider, as an example, the five-in-a-row board configuration at the right. All the moves can be encompassed in the 5 x 5 square shown. Thus, in the first step we represent this board configuration by a total of 26 integers, namely 5, 1, 3, 1, 2, 2, 2, 1, 2, 2, 2, 3, 1, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2. These 26 integers would then be encoded, using our earlier procedure, into a single positive integer. Similarly, then, moves, etc. in five-in-a-row could be encoded into integers. The question of whether a five-in-a-row configuration is o-winnable, or whether a move is a good move, can thus become a question about positive integers.

What is important in the above is not so much the details of the various encoding procedures. Rather, it is the following idea: Any finite body of information can be suitably encoded into a positive integer.

5. Procedure

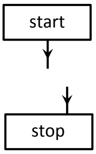
We now have available a standard input for our questions: We may take as the input an integer, secure in the conviction that we can always encode what we really want to ask about into such an integer. It is time now to standardize the procedures themselves.

The idea is to supply the technician with two things: a scratch pad, in which the technician can record various intermediate results for later use as well as the final answer; and a set of instructions, which will direct the technician in explicit detail what to do.

For the scratch pad, we shall use a number of bowls, each capable of holding an arbitrary number of beans (9, 1, 2, . . .). At any moment, a bowl must hold some specific number of beans (say, “13” – It cannot hold “an infinite number of beans”.) We may indicate the bowls by drawing them on paper, or, more easily, by giving each one a name, e.g., “Nancy”,, “C”, “137”, or “⋆ ”. It is only necessary that we (actually, the technician) be able to identify and distinguish these names. Any given procedure may use but a finite number of bowls (so the technician can be guaranteed that, by hunting around, he can always find the appropriate bowl).

For the instructions, we shall proceed as follows. We first introduce seven basic instructions. We then string these together, to make more complicated sequences of instructions, by use of flow charts.

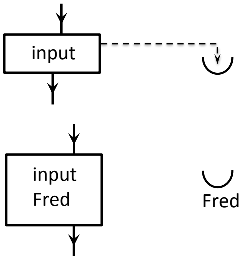
The seven basic instructions are the following:



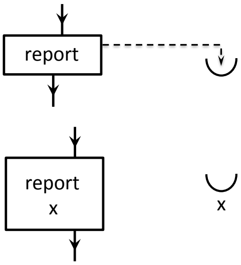
The instruction “start” has only a flow line emerging from it. The technician will always begin by finding, and then starting from, a “start” instruction.

The instruction “stop” has only a flow line entering into it. The technician, on reaching the instruction “stop”, stops.

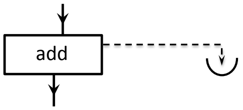
The instruction “input” has a single flow line entering into it, and single line emerging from it. In addition, this instruction must indicate one of the bowls comprising the scratch pad. This could be done, for example, by drawing a little dashed arrow to the bowl, or by giving the name of the bowl. On reaching the “input” instruction, the technician waits for us (who are overseeing all this) to place within the indicated bowl some number of beans of our choosing. Once this has been done, the technician continues. The role of this instruction is to allow us to supply information to the technician.



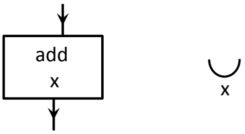
The instruction “report” has a single flow line entering into it, a single flow line emerging from it, and also indicates one of the bowls. On reaching the “report” instruction, the technician announces to us the number of beans then in the indicated bowl, and then continues. The role of this instruction is to allow the technician to supply information to us.

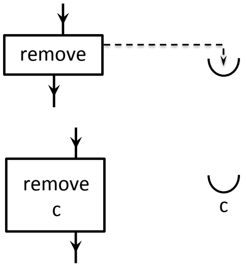


The instruction “add” has a single flow line entering into it, a single flow line entering into it, a single flow line emerging from it, and also indicates one of the bowls. On reaching the “add” instruction, the technician drops one additional bean into the indicated bowl, and then continues. This instruction, which will ultimately allow the technician to “calculate”, is automatic: We play no role in it.



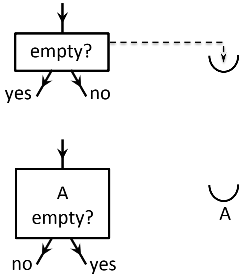
The instruction “remove” has a single flow entering into it, and also indicates one of the bowls. On reaching the “remove” instruction, the technician takes one bean from the indicated bowl (That is, if there is a bean in that bowl. If the bowl was empty, the technician leaves it empty.), and then continues.





The instruction “empty?” has a single flow line entering into it, and *two* flow lines emerging from it (these marked “yes” and “no”), and also indicates one of the bowls. On reaching the “empty?” instruction, the technician peers into the indicated bowl to see whether or not it contains any beans. If the bowl is empty, the technician continues along the flow line marked “yes”, if not, along the line marked “no”. The number of beans in the bowl is not thereby changed.

These, then, are the seven instruction: two to begin and end, two to communicate with us, two to calculate, and one to “make decisions”.



We next use these seven instructions as building blocks, composing more complicated tasks from them by means of flow charts. The rules for flow charts are the following:

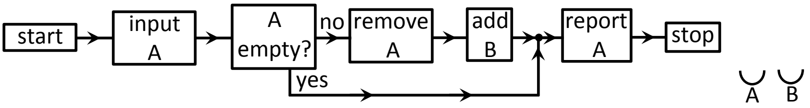
1. Every flow chart must have one and only one “start” instruction (for otherwise the technician wouldn’t know where to begin).

2. When several flow lines meet at a node, , any number of lines may come in, but one and only one flow line may emerge (for otherwise the technician, on reaching the node, would not know which way to go).

3. A flow line cannot just come to an end (for otherwise what is a technician to do?).

Let us also agree that initially, when the technician starts, all the bowls are to be empty.

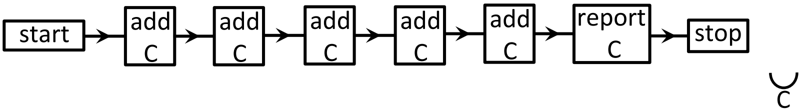
An example of how these instructions operate is provided by the following flow chart:



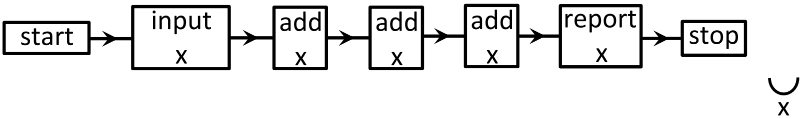
First note that each instruction has the correct flow lines entering and/or emerging from it, as well as appropriate bowl indications, and that the flow chart itself satisfies our rules. Here, the scratch pad has two bowls, named "A" and "B". A technician, following this flow chart, would proceed as follows. Beginning at "start", he would first ask us to place some number of beans in bowl "A". He would then test to see if this bowl is empty (i.e., if we put in zero beans). If not, he would remove a bean from bowl "A", and place it in bowl "B". Then, in either case, he would report to us the number of beans in bowl "A", and finally stop.

We now intend to argue in favor of the following assertion: *Such flow charts, with such instructions and such scratch pads, represent precisely our intuitive understanding of the word "procedure". That is, any task represented by such a flow chart would be regarded as a procedure; and any task we would regard as a procedure can be represented by a flow chart.* You probably would agree already with the first assertion (that any task represented by a flow chart would be regarded as a procedure), but are more skeptical of the converse. In any case, our argument will consist of giving various examples of flow-chart tasks. (It is almost impossible, by the way, to learn this subject passively. You must draw your own flow charts, invent your own, and play technician on your own.)

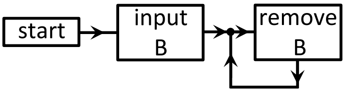
The flow chart below places five beans in bowl "C", and then reports the result, "5":

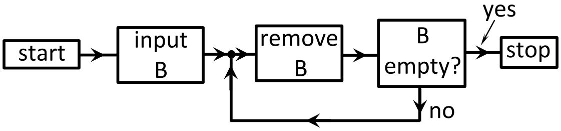


This flow chart inputs a number, adds three to it, and then reports the result:

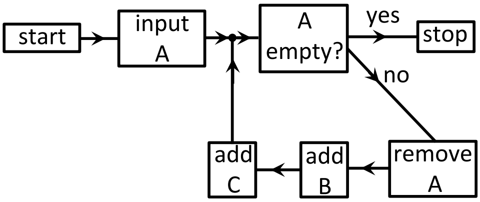


The first flow chart below empties the bowl B. However, this way of accomplishing the task is an inconvenient one, for it ties up a technician indefinitely. (Even after B is empty, the technician continues, forever, to try to remove additional beans.) A more convenient version is the second. We ask the technician to test B for emptiness after each bean-removal.

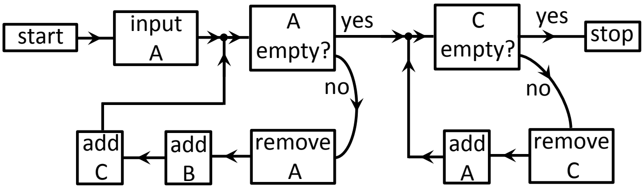




The flow chart below "copies" the contents of bowl A into bowls B and C, at the same time depleting A to empty.

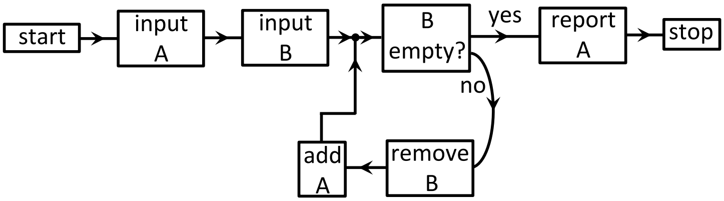


Thus, for example, if at the "input A" we were to place in A seven beans, then we would wind up, at "stop" with seven beans in each of B and C, and zero beans in A. How would one make a copy of A without emptying it? It is simple: One first copies A into B and C, while emptying A, as above; and then one copies C back into A:



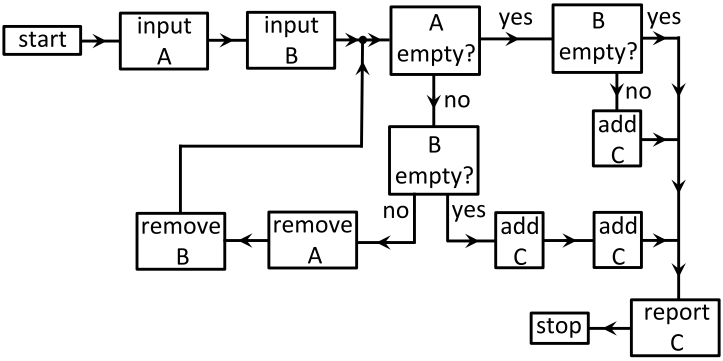
Similarly, we can make any number of copies of any bowl. This is most useful, for it allows us to employ our scratch pad to "keep records".

The flow chart below adds A and B, storing the sum in bowl A and emptying the bowl B:

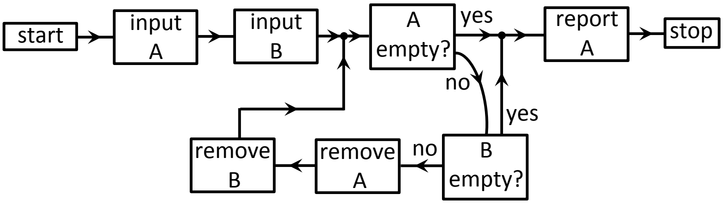


Suppose, however, that we wished to place the sum of the numbers in A and B in a new bowl, C say, while restoring A and B to their original numbers? This could be accomplished by combining the "add" and "copy" flow charts above: First copy A into C and B into D. Then add the numbers in C and D, with the sum appearing in C, as above.

The flow chart below takes two nonnegative integers, A and B, and determines whether A is greater than B, equal to B, or less than B:

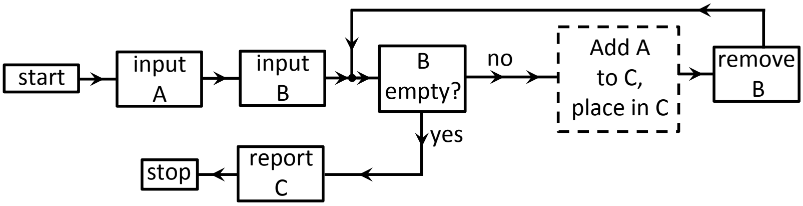


Here, the answer is encoded in the number of beans in bowl C, which is reported just before the end. The code is: 0 beans for A and B equal, 1 bean for A less than B, and 2 beans for A greater than B. Combining this with the "copy" flow chart, we could further arrange that the bowls A and B be restored to their original numbers at the end. Similar to this one is the flow chart that subtracts B from A (reporting zero if B is greater than A):

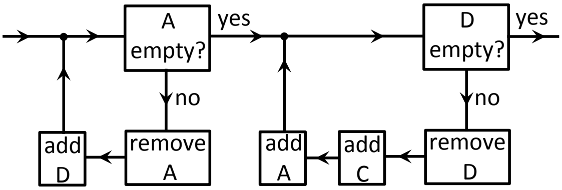


Suppose that one had wanted something something different reported when B is greater than A, say B minus A? This could be achieved by combining this flowchart with the one above: One would first employ the earlier flow chart to determine whether A is greater than, equal to, or lesser than B, and then require that different instructions be carried out depending on the outcome.

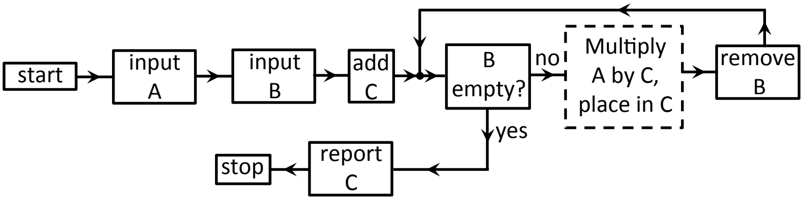
Multiplication is “repeated addition”. Based on this idea, we construct the following flow chart to multiply nonnegative integers A and B:



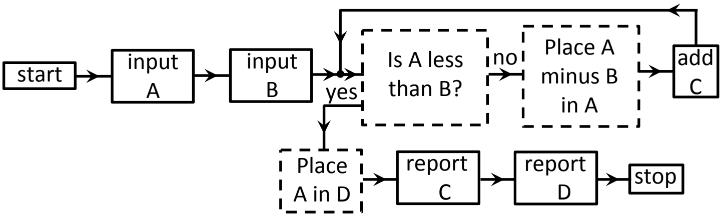
The dashed box in this flow chart is not one of our seven basic instructions. Rather, it stands for that combination of those instructions that carries out the following task: Cause C to contain at the end a number of beans equal to the sum of the numbers originally in A and C, and leave the number of beans in A unchanged. This is something we already know how to accomplish by means of our seven basic instructions. In more detail, then, the dashed box stands for the following:



Such abbreviations are a great convenience, of wish we shall make extensive use. In using such abbreviations, it is important that one i) indicate clearly within the dashed box what is to be done, and ii) be capable, if pressed, of actually doing this using the seven basic instructions. Let us also agree on the following: by “place. . . in bowl C” we mean “cause C at the end to contain exactly. . . beans”; and all bowls, unless otherwise indicated, are to be restored, within the dashed box, to their original numbers of beans. As a second example of such abbreviations, consider the following flow charts:



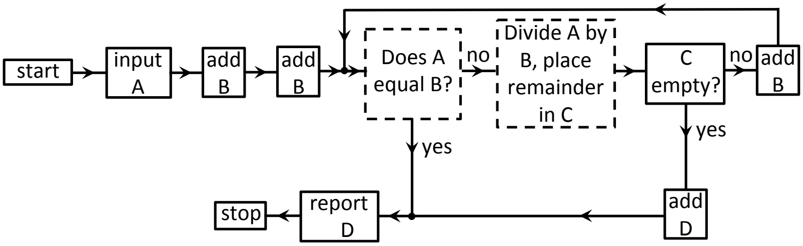
Here, the dashed box places in bowl C the product of the numbers of beans in A and C, restoring A to its original number. This is something we already know, from the discussion above, how to do. What does this flow diagram do? It raises A to the power B. (It is practically the same as the “multiplication” chart, but with multiplication now replacing addition.) Note that we must begin with one bean in bowl C.



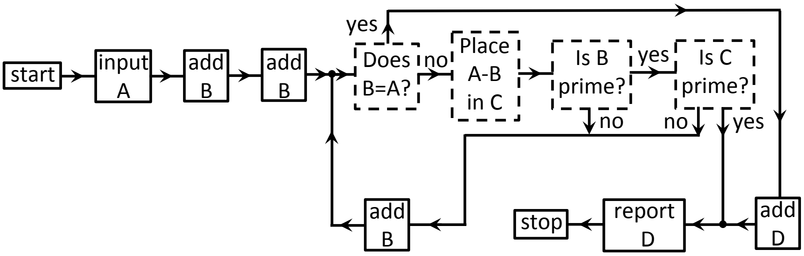
As a final example, we give a flow chart for division. Here, A is divided by B, with there reported first the quotient (in C), and then the remainder (in D).

Here, for the first dashed box there is to be determined whether or not A is less than B, and, with bowls A and B restored to their original values, the appropriate exit path taken. One could, by means of our basic instructions, carry this out. (One would use, basically, a modification of the second flow chart on page 28.) what will happen if, for "input B" there were placed zero beans in bowl B? How would one modify the flow chart to make something different happen?

We have now displayed flow charts to carry out certain bookkeeping operations on integers – copying and comparing – as well as the basic operations of arithmetic – adding, subtracting, multiplying, and dividing. More complicated operations are now constructed from these. Some examples follow.

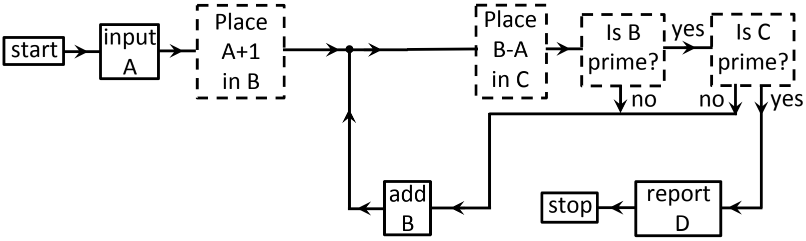


The flow chart above determines, provided A is at least two, whether or not A is a prime integer. This answer is encoded in the final report (which can, of course, only be an integer) as follows: 0 if prime, 1 if not prime. The flow chart operates, basically, by testing, for each number B from 2 up to one less than A, to see if it divides A without a remainder. What happens if A is zero or one?



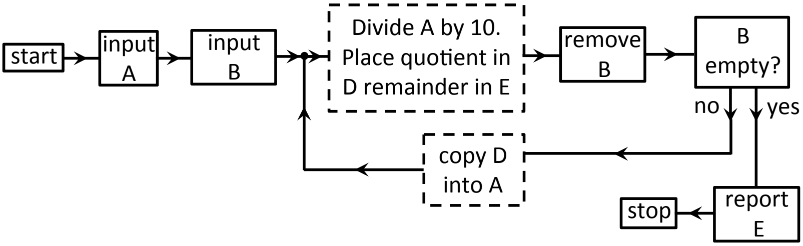
The next flow chart, given above, answers the questions of whether or not A is the sum of two primes.

The answer is reported as an integer, with “0” for yes and “1” for no. Alternatively, one could ask whether A can be written as the difference of two primes:



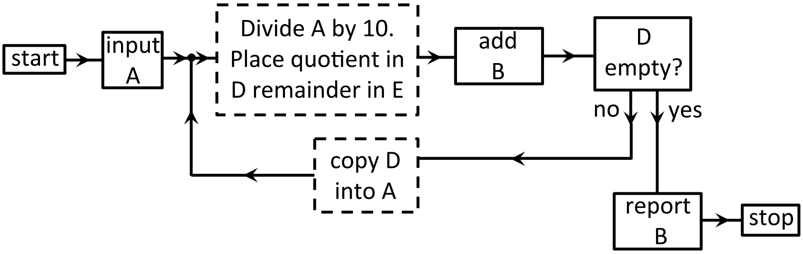
But notice an important difference between these two flow charts. In the first chart, the efforts of the technician cease in any case, with the report as to whether or not A is the sum of two primes. In the second chart, by contrast, the technician stops and reports only in the case in which A is found to be the difference of two primes. If not so found, the technician’s efforts continue forever. This difference reflects, of course, a difference between the two questions. There exists a procedure to answer the first, but not, as far as we know, the second.

The next flow chart finds, for B a positive integer, the Bth digit (counting from the right) of A:

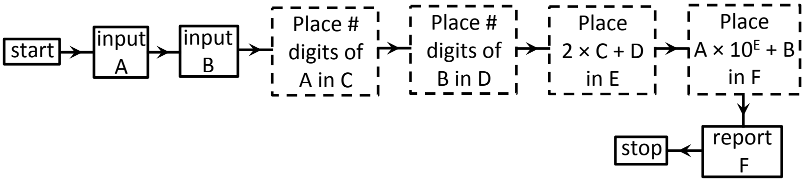


What does this flow chart report if B exceeds the number of digits in A?

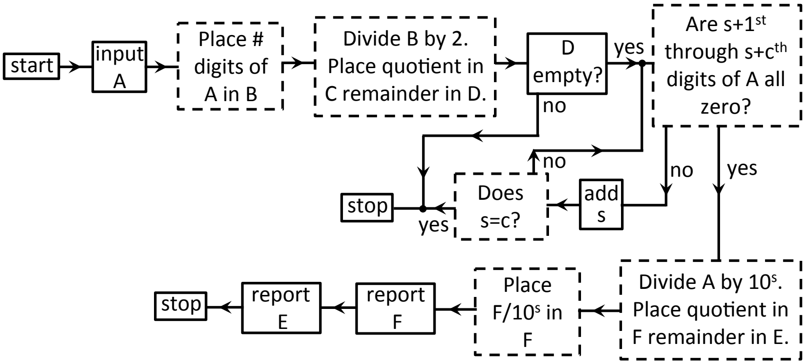
Similarly, we may have reported the total number of digits in A:



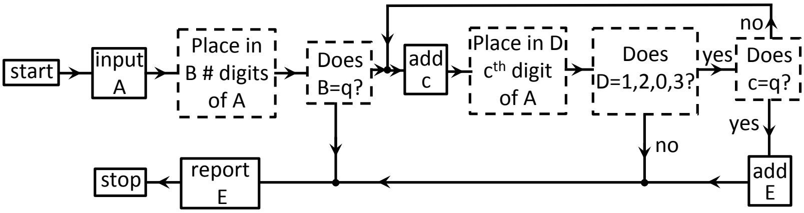
Once we can “count digits”, we can write the flow chart that takes positive integers A and B, encoding them, according to our prescription, into a single integer:



The next flow chart decodes:

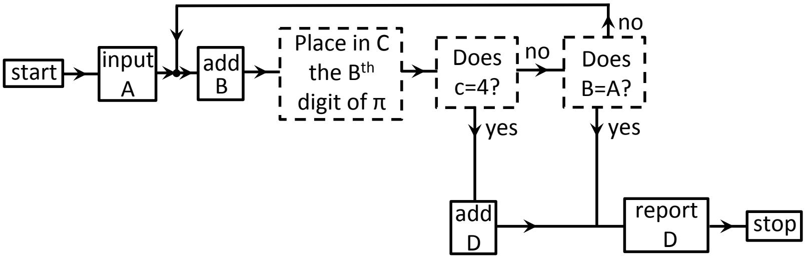

start
input A
Place # digits of A in B
Divide B by 2. Place quotient in C remainder in D.
D empty?
yes
no
Are s+1st through s+cth digits of A all zero?
no
yes
add s
Does s=c?
no
yes
stop
Divide A by 10s. Place quotient in F remainder in E.
Place F/10s in F
report F
report E
stop

In case A is not the encoding of two positive integers, the flow chart stops, without reporting anything. If A is such an encoding, those two numbers (in bowls F and E) are reported before stopping. The encodings of other types of information are treated similarly. For example, the flow chart below determines whether the integer A is the encoding of some tic-tac-toe board configuration:

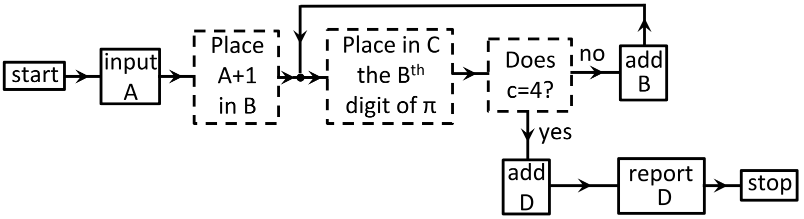


The number "1" is reported for yes, "0" for no.

The flow chart below determines whether or not there appears a "4" among the first A digits in the decimal expansion of π:

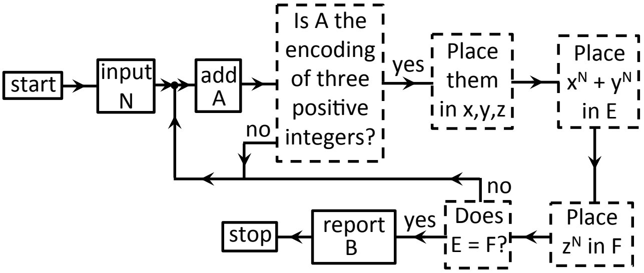


The answer, again, is reported as "1" for yes and "0" for no. Here, the dashed box marked "place in C the Bth digit of π" represents a certain (very complicated) flow chart that uses, for example, one of the series expansions for the number π. The details of this chart are not of interest here. Alternatively, one could ask whether "4" appears after the Ath digit:



But notice, in contrast to the previous one, that this flow chart can only report "yes" (i.e., "1"). If no digit "4" is found, then the technician will continue forever within the flow chart. Again, this behavior reflects the fact that we as yet have no procedure to answer this question, whereas we do have a procedure for the prior one.

As a final example, we give the flow chart that asks whether, given positive integer n, there exist positive integers x, y, and z with $x^n + y^n = z^n$. Here, we must "try various choices of x, y, and z", and one way of doing this is to "try choices of a single integer A, asking whether it is the encoding of three such integers".

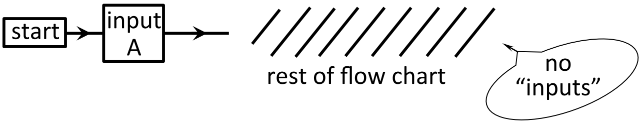


Here, the technician reports "0" and stops if such x, y, and z, are found, while the technician continues looking forever if none are ever found. This behaviour reflects the fact that we have currently no procedures for deciding, given n, whether or not there are positive integers x, y, and z with $x^n + y^n = z^n$. The best one can do is "look around for such x, y, and z", and indeed that is what our flow chart does.

It should by this point be clear that we could continue in this way indefinitely. It should be clear that such flow charts represent precisely our intuitive understanding of the word "procedure". It should be clear that any task represented by such a flow chart would be regarded as a procedure; and any task we would regard as a procedure can be represented by a flow chart.

6. Will the Technician Stop?

Let us consider now flow charts of the following special form: The start "start" is to be followed in succession by "input A" and then the rest of the flow chart. This "rest of the flow chart" is to have no further inputs, and is not to access the "input A". That is, we consider flow charts of the form given below.



This form guarantees, of course, that the technician, on executing the flow chart, will request from us an input one and only one time. Note that virtually every flow chart we have considered is already in, or can easily be cast into, this form. For those having no inputs, we merely add an "input A" right after the "start", and then ignore the A-value, thus generated, in the rest of the flow chart. For those having several inputs, we merely arrange to encode all the integers to be input into the single integer A. We then input this single A-value, and have the flow chart decode.

Let us consider now the following:

Big Question: Given a flow chart of the special form above, and given the nonnegative integer we shell provide at "input A", will the technician, on executing the flow chart, stop eventually, or continue forever?

This is clearly an infinite number of questions, one for each choice of flow chart and A-value. Given the choice of flow chart and of A-value, there certainly exists an answer to our question, for the result of the technician's executing the flow chart must certainly be either to stop or to continue forever. Of course, we may not be able to figure out what that answer is. (In particular, we cannot guarantee to answer the Big Question, for given flow chart and A-value, by merely hiring a technician to execute the flow chart, for were the answer to the Big Question in that case "continue forever", the technician would never be in a position to inform us that that is the answer.) (We might remark here that the reason for introducing the "special form" for flow charts on the previous page 37 was merely to standardize what is to be input to the flow chart, in order to make the Big Question easier to state.)

The Big Question (really, questions) is about flow charts. But, as we have seen, many questions within mathematics, as well as various questions about games and other things, can be reformulated to become questions about flow charts. Consider, as examples, the following:

1. Given positive integer n, is it the difference of two prime integers?

2. Given positive integer n, do there exist positive integers x, y, and z with $x^n + y^n = z^n$?

3. Does there exist integer n $\geq$ 3 and positive integers x, y, and z with $x^n + y^n = z^n$?

4. Do there exist ten consecutive "7" 's in the decimal expansion of π?

5. Given positive integer n, does there appear a "4" anywhere after the nth digit in the decimal expansion of π?

6. Given a tag game and initial string, will the game terminate?

7. Given the board configuration and whose move it is in five-in-a-row, is that configuration x-winnable?

For each of these questions, one could draw a flow chart that translates our original question into the question of whether or not execution of that flow chart eventually stops. In other words, each of these questions is a special case – for some particular choice of flow chart and A-value – of the Big Question. Indeed, virtually any question in mathematics of the form "Will a systematic search for blah-blah-blah be successful?" is some special case of the Big Question.

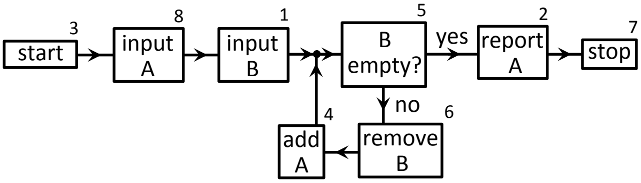
So, we could be the envy of our tag-playing friends, capture the

World five-in-a-row championship, and solve many of the heretofore unsolved problems in mathematics – if only we could answer the Big Question. But what does "answer the Big Question", as used here, mean? For any given instance – for any given choice of flow chart and A-value – one presumably *could* answer the question, merely by working on it hard enough. What is wanted here is, not the answer to the Big Question for some single instance, but rather the answer for every instance. In other words, what is wanted is a *procedure* to answer the Big Question. But what does "procedure" mean? It means a flow chart! Thus, we wish to construct a flow chart that will answer the Big Question. Such a flow chart would have to accept as inputs the "inputs" of the Big Question, namely flow charts and A-values. There is of course no problem with A-values. But in order that a flow chart accept flow charts as inputs, we must encode such charts as integers. This we now do.

Fix any flow chart. (This flow chart is to be already written in terms of the seven basic instructions, i.e., is to have all its "dashed boxes" already eliminated.) The first step is to count the total number of bowls utilized by this flow chart, and rename them using the integers from one up to this total number (in any order). The next step is to count the total number of boxes comprising this flow chart, and label them using the integers from one up to this total number (in any order). We now represent this flow chart by the following sequence of integers: number of bowls, number of boxes, integers referring to first box, integers referring to second box,. . . , integers referring to last box. Here, "integers referring to a box" are to be obtained using the following table:

start: 1, # of next box
stop: 2
input: 3, bowl #, # of next box
report: 4, bowl #, # of next box
add: 5, bowl #, # of next box
remove: 6, bowl #, # of next box
empty?: 7, bowl #, # of next box if "yes", # of next box if "no"

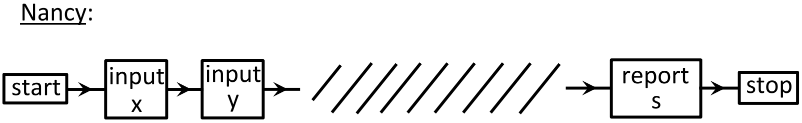
Thus, for example, there are three integers referring to a box containing a "remove" instruction, namely 6 (the integer code for "remove"), the number of the bowl from which the bean is to be removed, and the number of the box to which the technician is to proceed after executing this instruction. Consider, as an example, the following flow chart:



The first step is to number the bowls, say "1" for "A" and "2" for "B". The next step is to number the boxes from 1 to 8, say the numbering already given in the figure. Then this flow chart would be represented by the following sequence of integers: 2, 8, 3, 2, 5, 4, 1, 7, 1, 8, 5, 1, 5, 7, 2, 2, 6, 6, 2, 4, 2, 3, 1, 1. Translation: There are 2 bowls and 8 boxes. Box numbered 1 is an "input" (type 3) to bowl number 2 after which one proceeds to box number 5. Box numbered 2 is a "report" (type 4) from bowl number 1 after which one proceeds to box number 7.... Box numbered 8 is an "input" (type 3) to bowl number 1, after which one proceeds to box 1.

In this way, then, we may represent any flow chart by a sequence of positive integers. We now merely encode this sequence of integers into a single positive integer using our earlier method (for an indeterminant number of integers). In this way, then, we may encode any flow chart to a single positive integer. Of course, we can also decode, to obtain the chart from the integer: Given the positive integer, we first decode, as described earlier, to obtain the corresponding sequence of integers. From this sequence, we read off in turn the number of bowls, the number of boxes, and for each box, the type of instruction it represents, the bowl (if any) to which that box refers, and the box (s, if any) to which one proceeds after executing that instruction. But this information – what the boxes contain and how they are hooked together – is precisely what is needed to reconstruct the original flow chart.

Let us return now to the issue at hand: Whether there exists a procedure (read, flow chart) to answer the Big Question. Let us call such a flow chart – whose structure and operation we are about to describe – Nancy. Now, the Big Question has two "inputs": In order to ask the question, we must select, first, some flow chart (of our special form), and, second, some A-value. Hence, the flow chart Nancy should have two input instructions, allowing us to enter, first, the integer that encodes the selected flow chart, and, second, the selected A-value. The answer to the Big Question is either "stops" or "continues forever". Hence, the flow chart Nancy should have one report instruction, at which the answer may be reported (say, using the code "0" for "stops" and "1" for "continues forever"). That is, the flow chart Nancy should have the following form:



The middle portion should contain only the instructions "add", "remove", and "empty?" (for we do not any starting, stopping or communicating with us to take place there). So far, we have only arranged that Nancy have the correct inputs and outputs to match those of the Big Question. Now comes the important part: We must demand that Nancy correctly answer the Big Question. In more detail, we demand:

Let the flow chart Nancy be executed, and let there be entered, at "input X" the integer that encodes any flow chart of our special form, and, at "input Y", any integer. Then the execution of flow chart Nancy must always stop, there having been reported, at "report S", "0" if that flow chart (the one whose encoding was entered at X) with that A-value (the integer that was entered at Y) stops, and "1" if it continues forever.

So far, we have merely stated what it is we want the flow chart Nancy to do. We have certainly not issued any guarantee that there exists such a flow chart. So let us now, just to get an idea of what is involved, consider how one might go about trying to construct such a flow chart. One might think first of having Nancy merely execute the flow chart (the one whose encoding was entered at X) with the A-value (entered at Y). Indeed, it is not difficult to design a flow chart that will do precisely this. But, although we may construct such a flow chart, it would not do for Nancy. The problem is that, while it would correctly report “0” in the case of “stops”, it would be unable ever to report “1” for “continues forever”. Trying to construct a flow chart Nancy along these lines is analogous to trying to answer the Big Question merely by hiring a technician to execute the selected flow chart with the selected A-value.

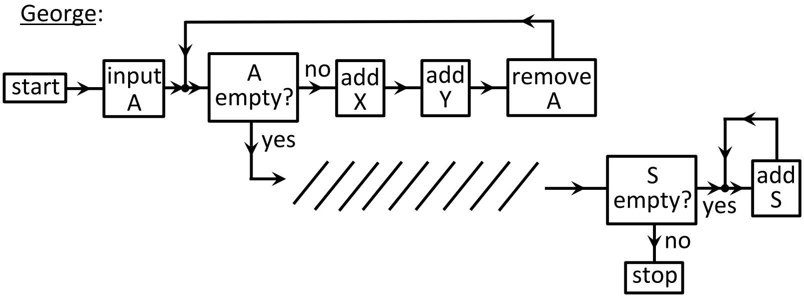
Clearly, the flow chart Nancy must, not merely execute the flow chart encoded at X, but rather examine the structure of that flow chart to determine, by some indirect argument, whether or not it (with the selected A-value) will stop. For example, the flow chart Nancy might first examine the flow chart encoded at X to determine whether or not it contains a “stop” instruction. If there is no “stop” instruction, then that flow chart, when executed, can clearly never stop. Hence, on so finding Nancy would set S equal to “1” (the code for “continues forever”) and exit at the instruction “report S”. Similarly, if the flow chart encoded at X contains no nodes (points where several flow lines meet), then execution of that flow chart must always stop. Hence, flow chart Nancy could check to see if the flow chart encoded at X has such nodes. If not, then Nancy would set S equal to “0” and exit. Here, then, are two examples of indirect arguments by which Nancy, on examining the structure of the flow chart encoded at X, might determine whether or not execution of that flow chart will ever stop. Clearly, much more sophisticated arguments than these two would have to be employed by Nancy. But one could in fact invent more sophisticated arguments showing that a flow chart will or will not stop, when executed, under various circumstances. Thus, it is at least conceivable that there could exist an argument so sophisticated that it could decide in *every case* (i.e., for every choice of flow chart of our special form and every choice of A-value), whether or not that flow chart, with that A-value, will stop. It is at least conceivable, in short, that there could exist a flow chart Nancy. On the other hand, one could equally well imagine that there exists no such Nancy.

Does there exist a flow chart Nancy? Is our career as a famous mathematician and world-champion five-in-a-row player just around the corner? Alas, it is not to be:

Theorem. There exists no flow chart Nancy.

Proof: Suppose for a moment that we were given a flow chart Nancy. (We shall obtain X- and Y- values for which this alleged “Nancy” in fact gives the wrong answer. From this it follows that no Nancy can exist.)

We now construct a new flow chart, which we call George, as follows:



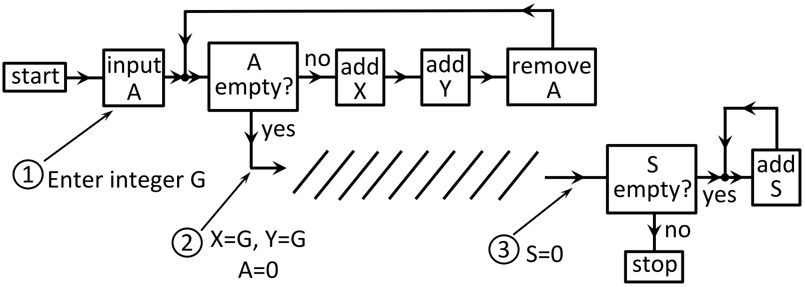
The first three boxes of Nancy, “start”, “input X”, “input Y”, are here replaced by six boxes, as shown; and the last two boxes of Nancy, “report S”, “stop”, are here replaced by three boxes as shown. But the entire reminder of the flow chart Nancy – the part indicated by / / / / / / / ; the part that does the real work – is left intact. The first three boxes of Nancy served merely to input values for X and Y. There are replaced in George by five boxes that input a single number A and then copy it into bowls X and Y. The last two boxes of Nancy served merely to report the value of S (either “0” or “1”) and stop. These are replaced in George by three boxes that cause George to continue forever in the case S is “0”, and stop otherwise. In short :

Flow chart George accepts a single nonnegative integer, copies it into bowls X and Y, and then has Nancy run with those X- and Y- values. If Nancy would have reported “0”, then George continues forever; while if Nancy would would have reported “1”, then George stops. But recall that reporting “0” is Nancy’s way of saying that the flow chart encoded at X, with A-value at Y, stops; while reporting “1” is her way of saying that it continues forever. Thus; *flow chart George does* (by stopping or continuing forever) *the opposite of what Nancy says* (about the flow chart encoded at X, with A-value at Y).

Thus, given flow chart Nancy (and we are supposing, for the moment, that we were given a Nancy) we could perfectly well construct flow chart George as indicated. And it would do what we said it does. Now George is merely some flow chart. Hence, we may encode it as a positive integer. Call this integer G, so G is merely some specific (although very large, I am sure) integer. We may also note that flow chart George is of our special form. That is, it begins with “start”, “input A”, and has no further inputs.

Now comes the key part of the argument. Let us now execute flow chart Nancy, and, at both “input X” and “input Y”, enter the integer G (the integer that encodes George). We claim that, whatever answer Nancy gives in this case, it must be the wrong answer.

Suppose, for example, that Nancy, when executed as above, gives value “0” at “report S”. What Nancy is predicting, then, is that flow chart George (the flow chart encoded at X), with A-value G (the integer entered at Y) will stop. So let us check this prediction of Nancy by simply running flow chart George, entering integer G at “input A”.



By position ② George will have completed his copying of A into X and Y. Hence, we shall at this position have G beans in each of bowls X and Y. Next, George runs Nancy with these X- and Y-values. There results, as position ③, S = 0 (for that is what we are supposing of Nancy, by the first sentence of this paragraph). Hence, George will continue forever (for he has S = 0 on entering the box “S empty?”). Thus, Nancy’s prediction – that flow charge George with A-value G will stop – is wrong, for when we actually run flow chart George entering A-value G we find that George continues forever.

But what if Nancy makes the other prediction? What if Nancy, when executed with integer G entered at both “input X” and “input Y” gives value “1” at “report S”? What if Nancy predicted that flow chart George, when run with A-value G, continues forever? Again, we may check this prediction of Nancy by simply running flow chart George, entering integer G at “input A”. Everything will be the same as before, until position ③, at which we shall now have S = 1. Hence, George will stop (for he will have S = 1 on entering the box “S empty?”). So, this prediction of Nancy would be wrong too.

To summarize, we have shown that, given any candidate for a flow chart Nancy, there is some choice of X- and Y-values (namely, X = G and Y = G, where G is the integer that encodes a certain flow chart George, which, in turn, is constructed from our candidate for Nancy) for which that alleged “Nancy” must give the wrong answer. We conclude that there exists no flow chart Nancy. (End of proof.)

This is a difficult and confusing proof. In order to be fully understood, it has to be read slowly and carefully – probably on several different occasions. But the idea of the proof – stripped of all the technical details – is quite simple. We suppose that we had a candidate for our desired flow chart, Nancy. We use this Nancy to construct flow chart George. George takes a single integer (for instance, 638), and proceeds to ask Nancy ”What happens if the flow chart encoded as 638 is run with A-value 638?” If Nancy answers “It stops.”, then George continues forever: and if Nancy answers “It continues forever.”, then George stops. That is, George *does* the opposite of what Nancy *says.* The problem arises when we use, instead of integer 638, the integer that encodes George himself, the integer we call G. For this case, George is asking Nancy “What happens if flow chart George is run with A-value G?”. But, since George (by design!) does the opposite of what Nancy says, either answer by Nancy to this question must be the wrong answer. In short, our candidate for Nancy does not work. Finally, since every candidate for a Nancy does not work, there can exist no Nancy.

Note that this proof is constructive: We actually *construct*, given a candidate for a Nancy, X- and Y-values for which that candidate must fail. Imagine that one gave as a test problem “Draw a flow chart Nancy.” This would of course be an easy problem to grade, for by the theorem (“There exists no flow chart Nancy.”) everyone receives a score of “0” on the problem. Nut then, after the tests are graded and returned, various students come up claiming that their Nancys actually work. There students are not interested in abstract proofs about existence or nonexistence of a Nancy: They want to know what is wrong with their individual Nancys. Fortunately, the proof of the theorem is such as to make it easy for the grader to prevail in these confrontations. A given student has a candidate for a Nancy. The grader merely takes that student's candidate for Nancy, constructs the corresponding George, determines the number G that encodes that George, and says to the student: "Try your Nancy out with X-value G and Y-value G". The student's candidate for Nancy will not work for these particular values. So, that student's score, "0", is justified. Of course, different students will have different candidates for Nancy, and so will receive different numbers (G) to try their candidates on. Thus, one student might be told by the grader "Try your Nancy out with X-value 31206 and Y-value 31206." Another student's flow chart might work for *these* X- and Y-values, but that student would be told "Try your Nancy out with X-value 4110002142 and Y-value 4110002142." Each student would be given values for which his or her particular flow chart would fail.

Note that there is no "paradox" here. (I have never been able to understand what this result has to do with the familiar self-reference paradoxes, with which it is often linked. "A barber shaves everyone who does not shave himself. So who shaves the barber?") Here, we merely state a theorem, and proceed to prove it.

The construction of flow chart George, by modifying a candidate for a Nancy, is rather delicate. The modification of the beginning of Nancy serves essentially to reduce the number of "input" instructions from two (in the case of Nancy) to one (in the case of George). The reason for this reduction is the following. Nancy deals only with flow charts "of our special form" (i.e., having "start", "input A", and no further inputs). But Nancy herself is not of this form, for she has two inputs.Thus, as things stand, Nancy cannot ask about herself. Flow chart George, on the other hand, is "practically Nancy", but, because of the modification of inputs, *is* of our special form. Thus, Nancy can ask about George, and so, indirectly, about Nancy herself. It is asking Nancy, indirectly, through George, about herself that makes the proof work.

Why are the other details of George as they are? Why does George copy the *same* integer into both X and Y? Why does George *do* the opposite of what Nancy says, as opposed, e.g., to *saying* the opposite of what Nancy says? Or, why not have George do the *same* as Nancy says? The easy answer to these questions is "if you construct George differently according to one of these proposals, then you do not get a proof of the theorem." After all, a proof in mathematics has only the obligation to be a proof. It does not have to be transparent, natural, or even interesting. I must confess that I do not have any deeper answers to these questions. The construction works.

We have agreed to take "procedure" to mean "flow chart". Thus, the meaning of the theorem is that there exists no procedure to determine, for any given flow chart of our special form and any given A-value, whether that flow chart with that A-value will stop or not. Imagine that it were your job to receive each day a flow chart (of our special form) and A-value and to determine whether that flow chart with that A-value will stop. Some would be easy; some would be hard. But, presumably, one could in every case – given sufficient effort and ingenuity – find the answer. But effort and ingenuity are hard work. One might look for an easier way. One might look for some procedure that could be followed in every case – some procedure that would be guaranteed in every case to produce the answer. The theorem guarantees that no such procedure can ever be found. There is, if you like, always a place for effort and ingenuity.

This situation should be contrasted with that of problems we have discussed earlier.

We have given examples of questions for which there is a procedure (Given positive integer n, is n prime?), and of questions for which we do not know whether or not there is a procedure (Given positive integer n, do there exist positive integers x, y, and z with $x^n + y^n = z^n$?). But we had not given earlier any example for which one could positively assert that there *is no* procedure. Here, from that theorem, we have our first such example.

There are more levels, than one might have expected naively, at which one can understand a question. No longer are there merely questions for which "We know that we can determine the answer." and those for which"We do not know whether or not we can determine the answer." Now, there are in addition questions for which "We know that we cannot determine the answer."

7. Appendix

Phy Sci 113 Sep 27, 1988

Due: Oct 4, 1988

Problem Set 1

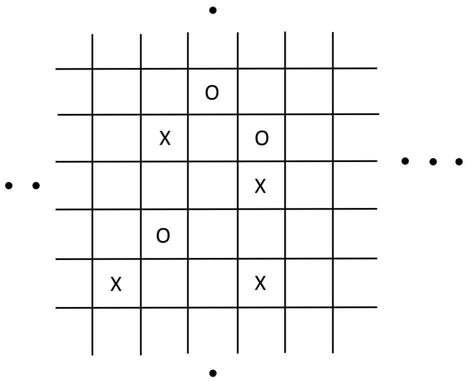
1. Learn to play tic-tac-toe expertly. (By "expertly" I mean such that yoou can always make the best possible move. Most of you probably know how to do this already. If you do not know the game, please ask, a friend.) Explain, in no more than a page, the procedure for expert play.

2. Nim is a game for two players. A number of beans is first counted out into a bowl. The players alternate turns, with each player, in his turn, taking either one or two beans from the bowl. The player taking the last beans wins. Learn to play nim expertly. Explain the strategy. (It is easier than it sounds! First try some games with a small number of beans.)

3. Learn to play checkers passably. (By "passably" I mean knowing the rules, and playing at a level of competence that comes from having played a few games. We shall adopt in checkers the following two rules: i) You must make a jump if one is available; and ii) if, during the course of the game, the board returns to exactly the same position for the third time, then that game is declared a draw.)

4. Five-in-a-row is a game played on a grid of vertical and horizontal lines. (The grid is "infinite", i.e., additional lines on the sides and top and bottom can be added as needed.) There are two players,

who alternate turns. Each player, in his turn, makes his mark (say, one player “x”, one “o”) on any unoccupied square. The first player to get five of his marks consecutively in a row (horizontally, vertically, or diagonally) wins. Learn to play five-in-a-row passably.

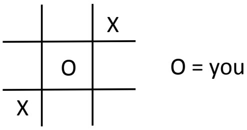


Phy Sci 113 Oct 4, 1988

Problem Set 1 — Solutions

1.

Rule 1. If you are able to win, i.e., to achieve three in a row, on that turn, do so.
Rule 2. If you are able to block your opponent's win, i.e., to take the third in a row in which he has two, do so.
Rule 3. If the center is available, take it.
Rule 4. If you hold the center only, and your opponent holds just two opposite corners, take one of the sides.



Rule 5. If a corner is available, take it.
Rule 6. Take any move.

The strategy is to move according to Rule 1 if you can. If you cannot then try to move according to Rule 2; if you cannot, then according to Rule 3; etc. Continue down the rules until you are able to make a move.

2. It is your move. Take the number of beans then in the bowl, divide by three, and find the remainder. (Thus, for example, if there are eleven beans in the bowl, then the remainder is two; if there are twenty-one, then the remainder is zero.)

If the remainder is zero, then take either one or two beans from the bowl.
If the remainder is one, then take one bean from the bowl.
If the remainder is two, then take two beans from the bowl.

In the first case, you will lose if your opponent plays expertly.
In the last two cases, you are assured of a win by this strategy.

Phy Sci 113 Oct 4, 1988
Due: Oct 11, 1988

Problem Set 2

1. Consider the tag game with just two symbols, a and b, and with the following rules:

A. If the leftmost symbol of the string is "a", then append "aa" to the right and remove the leftmost three symbols.

B. If the leftmost symbol of the string is "b", then append "bbab" to the right and remove the leftmost three symbols.

a. Show that if the initial string is of the form "axxaxxaxx... axx", where each "x" stands for either an "a" or a "b", then the tag game eventually terminates at the empty string.

b. Show that if the initial string is of the form axxbxxaxxbxx... axxbxx", then the tag game eventually cycles.

c. Try to find an initial string for which the tag game goes on indefinitely, without terminating or cycling. (Note: This may be difficult. I tried it for an hour or so, without success. So, if you have not found one after reasonable effort, just describe for your answer what methods you tried, and how they failed.)

2. Consider the tag game above, but with "a" leftmost append, instead of "aa", and "aba"; and with "b" leftmost append, instead of "bbab", "bbba". For this game, give the procedure for deciding, for any given initial string, whether the game eventually terminates at the empty string, cycles, or goes on indefinitely without terminating or cycling.

3. Select any two board games with which you are familiar, other than those discussed in class. For each of your games, discuss whether the problem of making expert moves is well-posed, whether it has a solution, and whether there is a procedure for expert play.

Remark: If, for the tag game of problem 1, you can figure out what happens with the initial string "bbbbbbbbbbbbbbbbbbbbb" then you will become famous, for this is an old, much-worked-on, and as yet unsolved problem.

Phy Sci 113 Oct 11, 1988

Problem Set 2 — Solutions

1.a. Suppose, e.g., that the initial string were "axxaxxaxxaxxaxx". Then the results of the first five moves would be:

axxaxxaxxaxxaa, axxaxxaxxaaaa, axxaxxaaaaaa,
axxaaaaaaaa, aaaaaaaaaa.

Thus, after five moves the string is all "a" 's. But now each further move will merely add two "a" 's to the right while deleting three from the left, i.e., each further move will again result in a string with all "a" 's, but with one fewer "a" for each move. Thus, successive moves from here will produce: aaaaaaaaa, aaaaaaaa, aaaaaaa, aaaaaa, aaaaa, aaaa, aaa, aa, a, (empty). Similarly, an initial string with n "axx" 's will, after n moves, be all "a" 's, after which it will shrink to the empty string.

b. The result of the first two moves on the string "axxbxxaxxbxx... axxbxx" are

bxxaxxbxx... axxbxxaa, axxbxx... axxbxxaabbab.

Thus, we obtain after two moves a string of exactly the same form as the original string, but with the "x" 's in the final six entries made explicit. (That is, instead of ending in "axxbxx", our string now ends in "aabbab".) Repeating this a number of times equal to the number of "axxbxx" 's in our original string, we end up with the string "aabbabaabbab... aabbab" (i.e., the string with each original "axxbxx" replaced by "aabbab"). But now two further moves on this string give the same string back. So, we have arrived at a cycle.

c. I still have not found such an example, and I am beginning to think that I am not going to find one soon. Here is just one possible strategy (which seems to fail) to search for an example. Suppose that one could find a string such that, after one has made enough moves to "use up" (i.e., totally delete) that string, what results is the original string again, but with some additional letters on the right. One might then hope that, as the game continues, one will again recover the original string, but with still more letters on the right, etc. Then, with any luck, one will be able to guarantee that the lengths of the string will increase without bound. But it seems difficult to implement this strategy. Here is a good-looking candidate: bxxbxxbx.

Here are the results of the first three moves: bxxbxbbab, bxbbabbbab, babbbabbbab. Note that we now have the form of the original string ("bxxbxxbx", here made explicit, to "babbbabb") with some additional letters (namely, "bab") on the right. It looks promising. But let us continue the game from here: bbabbbabbbab, bbbabbbabbbab, abbbabbbabbbab, babbbabbbabaa. It looks very promising at this point! We once again have our original string, but now with "babaa" on the right. But if we continue to play from here, things break down: bbabbbabaabbab, bbbabaabbabbbab, abaabbabbbabbbab, abbabbbabbbabaa, abbbabbbabbaaaa, babbbabbaaaaaa. I just do not see how, from here, to guarantee that we will continue, after every few moves, to recover our original string, plus more and more extra letters on the right.

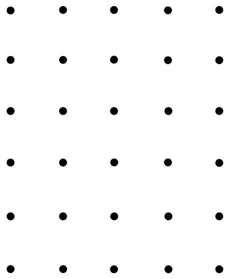
2. First note that, under these rules, no game ever terminate at the empty string. Indeed, each step in the game either leaves the length of the string the same (in the case the leftmost letter is "a"), or increases the length of the string by one (leftmost letter "b"). Thus, we have only to decide whether the game cycles, or goes on to ever longer strings.

If, during the course of a game, a "b" were ever hit (i.e., were the leftmost letter) then we would append "bbba" on the right. But these three "b" 's in a row assure that one of them would be hit eventually (since, at each step, just three are removed from the left). This would result in three more "b" 's on the right, and so in another "b" hit, etc. We conclude: If, in the course of a game, a"b" is ever hit, then the game will go on to ever longer strings (since, as we have just seen, we must then continue to hit "b" 's). But clearly, if no "b" is ever hit, then the game must cycle (for then only "a" 's would be hit, and so the string length must remain the same).

Thus, there are just three candidates for games that cycle (i.e., those for which only "a" 's are hit), namely those with initial strings of the form "axxaxx... axx", "axxaxx... axxax", "axxaxx... axxa". Playing each of these initial strings for one step, we obtain, respectively, "axx... axxaba", "axx... axxaxaba, and "axx... axxaaba". We see that the first and third, but not the second, continue to be of our form. We thus obtain:

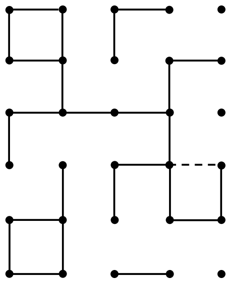
Answer: All initial strings of the form "axxaxx... axx" or "axxaxx... axxa" lead to games that cycle. All others continue to ever longer strings. No game terminates.

3. For the game of bridge, the problem is not well-posed in our sense. In the first place, there is concealed information (in that the hands are not shown to others), and so there is no board configurations giving the entire information about the status of the game. In this sense, bridge is more like poker. A further complication is that, in bridge, there are four players, acting in pairs as teams. Thus, a "player" in our sense would be an entire team, and one would have to allow the "players" to have two stores of information (those possessed by the two members of that team), such that information could not readily be passed back and fourth. Because of these complications, it is not even clear (to me, at least) that there exists "expert play" in bridge using probabilities for various moves (as there is, e.g., in poker). In any case, bridge is very far from the types of games we have in mind.



Consider, as a second example, the pencil-and-paper game "boxes". One begins with a finite grid of points on a piece of paper. There are two players, and each in his turn is allowed to connect two adjacent points by a horizontal or vertical line. Thus, the figure on the next page illustrates a typical board configuration. The dashed line indicates a move whose turn it is.

If, in your move, you complete the fourth side of a "box" (one of the little squares having four of the points as corners), then you are awarded one point, and you move again. Thus, in the example at the right, the person about to make the dashed-line move is about to receive one point (and to take another move). At the end (when all lines have been drawn in), the player with the most points wins the game.



This game, I claim, satisfies all of our criteria. There is a board with configurations giving all information about the game. There are two players, who alternate turns, and make certain allowed moves. There is a win, lose, and draw. (To make this clearer, one should have the players mark (with an "x" or "o") squares they have completed. Then the "board configurations", would include these marks on the squares. The "x-win configurations", then, would be those in which the entire board is filled in, and "x" has more squares than "o".

We conclude that, for the game of boxes, the problem of expert play is well-posed, and there does exist an expert strategy. It is not clear whether or not there exists a procedure for expert play. (Certainly there does if the size of the initial grid is fixed, for then there are but a finite number of board configurations. But for arbitrary games, there is no finite number.)

Do you think there is a procedure for expert play? What is your guess?

Phy Sci 113 Oct 11, 1988
Due: Oct 18, 1988

Problem Set 3

1. In each of the tic-tac-toe configurations below,

O	X	
	O	

		X
	O	

	O	X

O	O	X

it is x's move. For each, decide whether that configuration is x-winnable, o-winnable, or drawable, and give a good move for x.

2. Show that, for a game within our framework, if it is x's turn and the board configuration is drawable, then there exists a move for x that results in a drawable board configuration.

3. Given a positive integer n,

a. is it the sum of any number of prime numbers?
b. is it the square of a prime number?
c. is it any power of a prime number?
d. is it the sum of two odd integers?
e. is it the difference of two odd integers?

For each of these questions, determine whether or not there is a procedure for answering the question. If there is a procedure, give it.

4. Given positive integer n, give explicitly the procedure for deciding whether or not it represents an "x-wins" configuration in tic-tac-toe. (Make sure that your procedure really is that – something that can be followed mechanically.)

Phy Sci 113 Oct 19, 1988

Problem Set 3 — Solutions

1. Label the squares on a tic-tac-toe board as shown below.

1	2	3
4	5	6
7	8	9

a. This configuration is o-winnable. Thus, every move for x results in an o-winnable configuration, and so every move for x is a good move.

b. This configuration is drawable. It turns out that every possible move for x results in a drawable configuration, and so every move for x is a good move.

c. This configuration is drawbale. A move by x to 4 results in an o-winnable configuration, while all other moves – to 1, 2, 3, 7, 8, or 9 – result in a drawable configurations. Hence, any of these six moves is a good move.

d. This configuration is x-winnable. Moves to 6 or 9 result in an x-winnable configuration; to 5 or 8 in a drawable configuration; and to 4 and 7 in an o-winnable configuration. Hence, the good moves are to either 6 or 9.

2. The board configuration is drawable, and it is x's turn. There can be no move for x that results in an x-winnable configuration, for were there such a move then, by the rules that define x-winnable, the original configuration would already have been x-winnable. So, every move for x results either in a drawable or an o-winnable configuration. Were it the case that every move results in an o-winnable configuration, then, by the rules that define o-winnable, the original configuration would already have been o-winnable. Hence, there must be some move for x that results in a drawable configuration.

3.a. There does exist a procedure: "If the integer is 1, say 'no'; otherwise say 'yes'." (Every integer 2 or greater is the sum of some number of prime integers. Indeed, every such integer is the sum of some number of 2's and 3's.)

b. There does exist a procedure. Suitable instructions might direct that the technician take in turn each integer from 2 up to the given integer. First, test to see if it is prime, and then test to see if its square is the given integer. If in any case both tests are positive, say 'yes'; if none is found by the time you reach the given integer, say 'no'.

c. There does exist a procedure. Let the given integer be n. For each integer from 2 up to the given integer, first test to see if it is prime. Then begin raising that integer to various powers – first, second, and so on, continuing until the result exceeds n. Test to see whether the result ever equals n. If in any case both tests are positive, say 'yes'; if none is found by the time you reach n, say 'no'.

d. There does exist a procedure. Say 'yes' if the integer is even, and 'no' if it is not.

e. There does exist a procedure. Say 'yes' if the integer is even, and 'no' if it is not.

4. Say 'yes' if the given integer is one of the forms lxxlxxlxx, xlxxlxxlx, xxlxxlxxl, lllxxxxxx, xxxlllxxx, xxxxxxlll, lxxxlxxxl, xxlxlxlxx; and 'no' otherwise. Here, "x" can stand for any digit. (Note that we have, above, merely listed all possibilities for three x's in a row.) Remark: There is of course also a procedure to decide whether a given configuration is x-winnable, but that is far more complicated than this one. Basically, one would have to have the technician reconstruct the tic-tac-toe board, then play out the game using our earlier expert strategy, find out who wins, and answer accordingly.

Phy Sci 113

Oct 18, 1988
Due: Oct 25, 1988

Problem Set 4

1. Given positive integer n, is it the encoding of three positive integers, x, y, and z, such that x + y = z? Answer this question for each of the following n:

a. 30000000150008

b. 1300000000000000200000000213

c. 130000000000000200000000213

2. Is there a procedure for answering?

a. Given positive integer n, do ten 7's occur in a row in the first n digits in the decimal expansion of π?

b. Given positive integer n, is both n a prime and there exist positive integers x, y, and with $x^n + y^n = z^n$?

3. Encode the four positive integers 13, 2, 11, 100 into a single integer.

Phy Sci 113 Oct 25, 1988

Problem Set 4 — Solutions

1. a. No, The given integer is indeed the encoding of 3, 15, and 8. But it is not the case that 3 + 15 = 8.

b. Yes. The given integer is the encoding of 13, 200, and 213. It is the case that 13 + 200 = 213.

c. No. The given integer is not the encoding of any three positive integers. Indeed, it has an odd number (27) of digits.

2. a. There is a procedure. Use any of the known procedures for computing the first n digits of π, then inspect those digits to see if anywhere there occur ten "7" 's in a row.

b. We do not know whether or not there is a procedure. If the given n is not a prime, then clearly the answer is "no". But what if n is a prime? Then we do not have, offhand, any way to tell whether or not there exist x, y, and z with $x^n + y^n = z^n$. Thus, at least as the things stand, we have no procedure.

3. Encoding 11 and 100 yields 1100000100. Encoding 2 and this yields 2000000000001100000100. Encoding 13 and this yields 130000000000000000000000002000000000001100000100. This last monstrosity is the desired answer.

Phy Sci 113

Oct 25, 1988
Due: Nov 1, 1988

Problem Set 5

1. To specify a tag game, one must tell what is the alphabet, what is to be appended on the right according to which letter is on the left, how many leftmost letters are to be deleted, and what is the initial string. Encode all this information in a single positive integer.

2. Given positive integer n, do there occur n or more consecutive “7” ’s anywhere in the decimal expansion of π? Is there a procedure for answering this question? (Note: This is a *tricky* problem. It is worth thinking about for a while.)

Phy Sci 113 Nov 1, 1988
Due: Nov 8, 1988

Problem Set 5 — Solutions

1. There are of course many ways of encoding this information. Here is one. Given the tag game and initial string, first represent it as a sequence of integers, as follows:
total number of letters in the alphabet (we represent these letters by the numbers from 1 up to this total), number of letters in the string to be appended on the right if the first letter appears on the left, the successive letters (represented as numbers) in this string, number of letters in the string to be appended on the right if the second letter appears on the left, the successive letters in this string,. . . (up to these things for the last letter in the alphabet), number of letters from the left to be deleted in each turn, letters (represented as numbers) in the initial string.
Thus, for example, the tag game of the first test, with initial string "abbaa", would be represented by the following sequence of numbers: 2, 2, 2, 2, 3, 1, 1, 1, 3, 1, 2, 2, 1, 1. Translation: There are $\underline{2}$ letters in the alphabet. If the first (a) appears on the right, append the $\underline{2}$-letter string consisting of "bb" ($\underline{2}$ and $\underline{2}$). If the second (b) appears on the right, append the $\underline{3}$-letter string consisting of "aaa" ($\underline{1}$ and $\underline{1}$ and $\underline{1}$). Remove each time $\underline{3}$ letters from the left. The initial string is "abbaa" ($\underline{1}$ and $\underline{2}$ and $\underline{2}$ and $\underline{1}$ and $\underline{1}$).

Thus, we have so far represented a tag game, with initial string, as some finite sequence of integers. We now merely encode these into a single integer as was shown in class. (Here, of course, we must use "encoding an indeterminate number of integers", because we do not know beforehand how many there will be.) It is clear that the final integer can be decoded to find the tag rules and initial string.

2. (*Note:* I only put this problem on because it reflects a point I wanted to make, but for which I did not wish to use more class time. I did not really expect anyone to get the answer, below, but rather wanted to get you thinking about it in preparation for this point.)

There does exist a procedure to answer this question! To see this, let us consider a few possibilities.

Suppose that there are sequences of consecutive “7” ’s of arbitrary length in the expansion. (This means, in more detail, that given any positive integer there is some sequence of consecutive “7” ’s in the expansion of length larger than this integer.) Given this supposition, the answer to our question is “yes” no matter what n is (for there are such sequence longer than any given n). So, in this case there does indeed exist a procedure to answer the question, namely “say yes”.

Suppose that the longest sequence of consecutive “7” ’s in the expansion consists, say, of 188 “7” ’s. In this case, there is also a procedure to answer the question, namely “Say ’yes’ if n is less than or equal to 188; and ‘no’ if n is 189 or greater.”

Suppose the longest sequence of consecutive “7” ’s in the expansion consists of just 5 “7” ’s. Then there is again a procedure: “Say ’yes’ if n is less than or equal to 5; and ‘no’ if n is 6 or greater.”

To summarize, then, there is in every case a procedure. If there is no longest sequence of consecutive “7” ’s, then the first case above applies, with that procedure; if there is a longest sequence, then one of the other cases applies. We conclude: There does exist a procedure to answer this question.

But this is another example in which, although we know that a procedure *exists* (here, because we can list a bunch of procedures and be sure that *one* of them is the right one), we do not know which procedure is right. It is quite similar to those examples of “yes-or-no” questions in class, in which we knew that one of two procedures (“say ‘yes’ ” and “say ‘no’ ”) was correct, but not which. The only difference is that, here there are an infinite number of procedures in our list. Again, one is correct, but we do not know which one.

This example very well illustrates the point that “not being able to find a procedure offhand” is quite different from “there is no procedure”. Here, it turns out that there is a procedure, but it is just somewhat subtle.

Phy Sci 113 Nov 1, 1988

Due: Nov 8, 1988

Problem Set 6

Note: There are three skills to be acquired: i) how, given a simple procedure, to write a flow chart for it, ii) how, given a more complicated procedure, to explain clearly how one could, at least in principle, write a flow chart for it, and iii) how, given a flow chart, to figure out what procedure it performs. These skills can be mastered, in my opinion, only by playing around with these things on your own. The problems below are intended only to get you started in this.

1. Write a flow chart that accepts two nonnegative integers A and B, divides A by B (provided B is not zero), and reports the quotient and remainder.

2. Write a flow chart that accept two nonnegative integers A and C, and reports A − C if A is greater than or equal to C, and C − A otherwise.

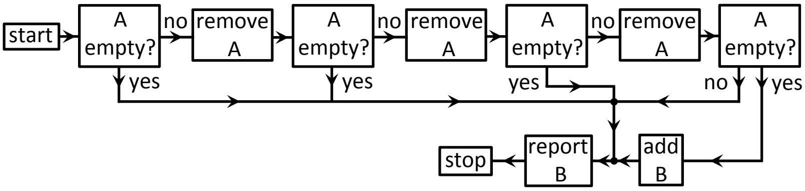
3. Write a flow chart that accepts three nonnegative integers X, Y, and Z, and reports the largest.

4. Argue that there exists a flow chart that accepts a nonnegative integer A and reports the number of digits in A.

5. Argue that there exists a flow chart that accepts a nonnegative integer A and reports whether or not A is the encoding of two other positive integers.

6. Argue that there exists a flow chart that accepts positive integers P and Q, and reports the result of our encoding of these two integers.

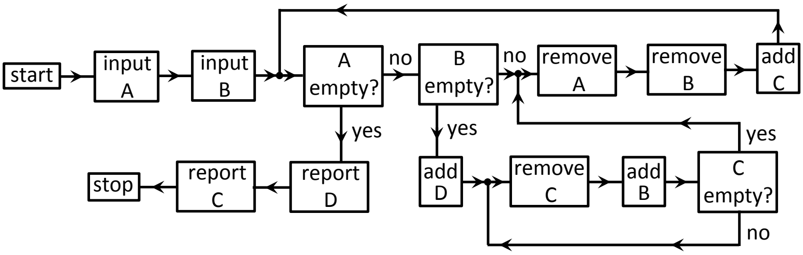
7. What does the flow chart below do?


start
A empty?
no
remove A
A empty?
no
remove A
A empty?
no
remove A
A empty?
yes
yes
yes
no
yes
stop
report B
add B

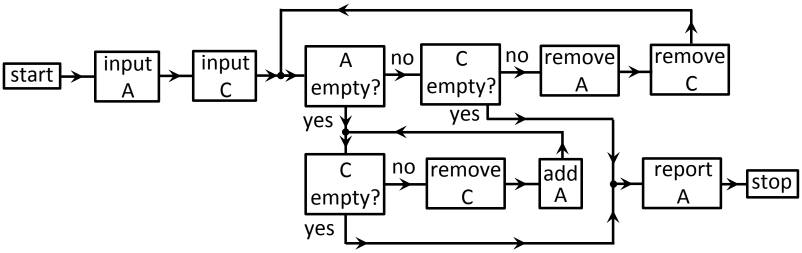
Phy Sci 113 Nov 8, 1988

Problem Set 6 — Solutions

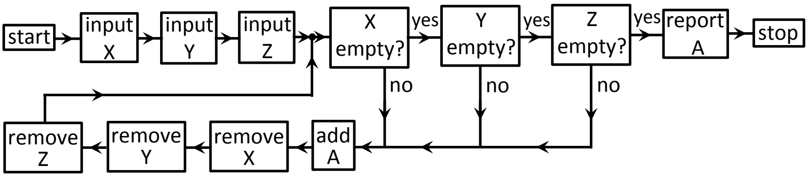
1.



2.



3.



4, 5, 6. These three problems were done in class, and also are done in the notes.

7. As the flow chart stands, the answer is "Always report zero." Correcting the flow chart by inserting the extra box "input A" after

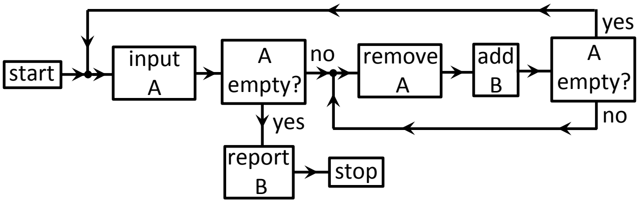
the first box (as I asked you, in class, to do), the flow chart answers the questions “Is the number A three, or not?” The result is reported as “1” if the answer is yes, and“0” if no.

Phy Sci 113 Nov 8, 1988
Due: Nov 15, 1988

Problem Set 7

1. Draw a flow chart that accepts as input three nonnegative integers, and reports which of these (first, second, or third) is the largest.

2. What does the flow chart below do?



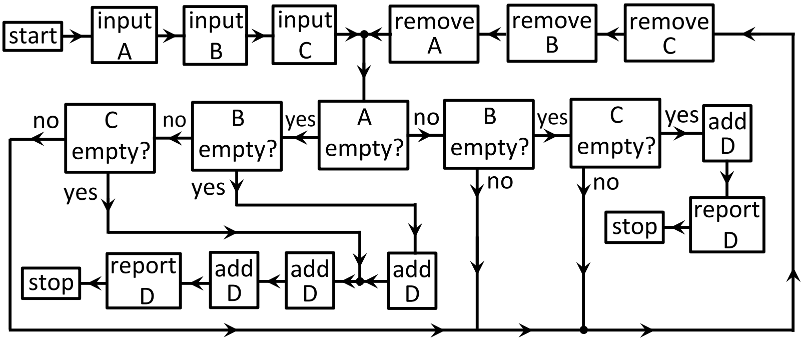
3. Draw a flow chart that accepts as input a nonnegative integer, and reports whether or not that integer is the largest prime.

4. Argue that there exists a flow chart that accepts as input a nonnegative integer, and reports whether or not that integer is equal to the sum of the squares of two integers.

Phy Sci 113 Nov 15, 1988

Problem Set 7 — Solutions

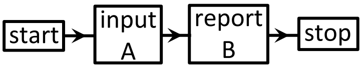
1.



This flow chart (which admittedly, is not very elegant) works provided there *is* a largest (as opposed to two- or three-way tie for largest).

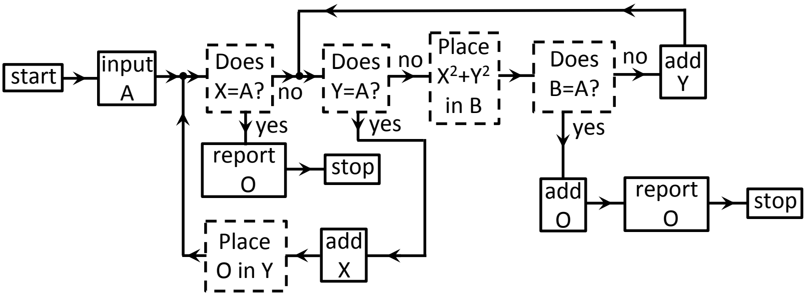
2. The user is continually asked to input numbers (a running total of which is kept in bowl B). As soon as "0" is input, this total is reported and the execution stops. In short, this flow chart is an adding machine?, with "0" the signal for for "give me the sum".

3.



Here, "0" is reported for "is not the largest prime", and "1" for "is the largest prime".

4.



Here, a “0” is reported for “no”, and “1” for “yes”.

Phy Sci 113

Nov 15, 1988
Due: Nov 22, 1988

Problem Set 8

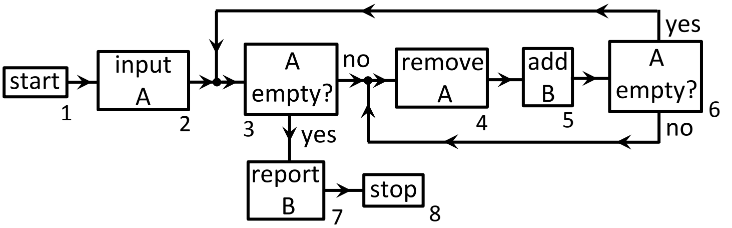
1. Translate the flow chart of problem 2, Set 7 into a sequence of integers. If this were to be encoded into a single integer, approximately how many digits would that integer contain?

2. Draw the flow chart represented by the following sequence of integer: 3, 7, 5, 2, 5, 2, 1, 4, 3, 1, 7, 5, 3, 7, 6, 1, 1, 7, 1, 2, 6.

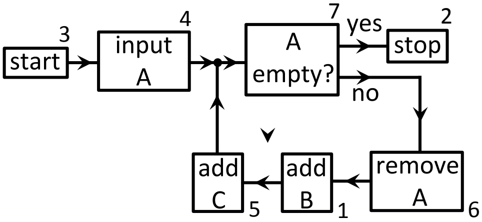
Phy Sci 113 Nov 22, 1988

Problem Set 8 — Solutions

1. 2, 8, 1, 2, 3, 1, 3, 7, 1, 7, 4, 6, 1, 5, 5, 2, 6, 7, 1, 2, 4, 4, 2, 8, 2. Here, we are numbering the bowls with "1" for "A", and "2" for "B". We have numbered the boxes as shown below:



2.



The bowls "A", "B" and "C" are numbered "1", "2", and "3", respectively. This flow chart, by the way, is from page 27 of the notes.

1. (continued) Each time we include one more number in our encoding (to a single integer), we approximately double the number of digits (actually, a bit more than double). Here, there are 26 numbers to be encoded (including the first number, "25", the number of numbers). Hence, there will be over 2^{26} digits. This is more that 60,000,000 digits!

Phy Sci 113 Oct 25, 1988

Test 1

Please write your name on every page. Please answer and *explain* your answer *briefly.*

Name:

1. (20 points)

a. A tic-tac-toe game has just begun, and it is x's turn to make the first move. Is the board configuration x-winnable, drawable, or o-winnable? Explain briefly.

b. A game within our general framework is underway, and the board is in some configuration. It is o's turn, and there are exactly eleven moves available to o. Six will result in an x-winnable configuration, three in a drawable configuration, and two in an o-winnable configuration. Is the present configuration of the board x-winnable, drawable, or o-winnable, or is there insufficient information to tell? Explain *briefly.*

2. (16 points) Give any prescription to encode an arbitrary integer (i.e., which may be positive or negative) as a positive integer.

3. (16 points) Consider the tag game with “a” and “b”, and with the following rules: For “a” leftmost, append “bb” on the right, and for “b” leftmost append “aaa” on the right; and in both cases delete the leftmost three letters. Give a procedure to determine, given the initial string, whether the game will ultimately terminate, cycle, or go on forever without terminating or cycling.

4. (30 points) You are to be given positive integer n. For each of the question below, state whether there exists a procedure to answer the question, there does not exist such a procedure, or you do not know whether or not there is a procedure. Explain *briefly.*

a. Has there occurred a "4" by the nth digit in the decimal expansion of π?

b. Does there occur a "4" anyplace after the nth digit in the decimal expansion of π?

c. Is n either the result of encoding two integers, or the difference of two primes?

Phy Sci 113 Oct 25, 1988

Test 1 — Solutions

1.a. The configuration is drawable. That is, it is not possible for x, absent a blunder by o, to be assured of a win; nor is it possible for o, absent a blunder by x. In other words, with expert play on both sided the game will certainly be a draw.

b. The configuration is o-winnable. Recall that if it is o's turn, and there is *any* move for o that results in an o-winnable configuration, then the present configuration is o-winnable. But that is the situation here.

2. There are many such prescriptions. For example, write out the digits of the integer, and affix a "1" on the right if it is positive, and a "2" if it is negative. Or, as a second prescription, ignore the sign the integer, double it, and then add one if the original integer was negative.

3. The procedure is: "Say, no matter what is the given initial string, 'It will terminate'." Clearly, every game will either cycle or terminate (since one adds at most three letters to the right, while deleting three, and so the string-length can never grow). The only possibility for cycling, then, would be if only "b" 's are hit. But a "b" hit results in appending "aaa", which in turn assures that eventually an "a" will be hit. Hence, the string must ultimately shrink to nothing.

4. a. There is a procedure: "Say 'no' if n is one, and 'yes' otherwise." This works because the digit of π ($= 3.14159\ldots$) is a "4", and so the answer is 'yes' for n two or greater, but, clearly, 'no' for n = 1.

b. We do not know whether or not there exists a procedure. There is no obvious way, given n, to find the answer. Thus, for example, one could not simply try out every digit past the n^{th} in the decimal expansion, for there are an infinite number of such digits.

c. We do not know whether or not there exists a procedure. Of course, if n is the result of encoding two integers, we say 'yes'. But what if n is not? Then we know offhand of no way, by some finite procedure, to determine whether or not n is the difference of two primes.

Phy Sci 113 Oct 27, 1988

Test 1 — Comments

This test was too easy for (or, if you prefer the other way of saying it, the depth of your understanding exceeded my already very high expectations). It is not that the average (56, out of a maximum 82) was so high – although it was somewhat higher than I expected – but rather that there were a fair number of people who were just not challenged by the test. This is clear from your comments, and also from the grade distribution:

Score	Nmbr receiving	Score	Numbr receiving
10-19	1	50-59	31
20-29	6	60-69	46
30-39	9	70-79	30
40-49	24	80-89	11

What I should have done here – and will try to do next time – is including at least one very hard problem, so those who are having no trouble with the material will have something to chew on. Also, this problem should in part take care of itself, for we are now, moving into somewhat meatier material.

Comments on individual problems:

1.a. This was intended as a "throw-in" problem, to get people relaxed, and, indeed, very few had much trouble with it. A few did not realize that either player in tic-tac-toe can, from the beginning, force a draw. A board configuration must be *exactly one* of x-winnable, o-winnable, or drawable – it cannot be two or more of these.

b. The status – x-winnable, o-winnable, drawable – is a property *only* of the board configuration (and whose turn it is). It does not depend on whether people play expertly, badly, or not at all. (Of course, the definitions of these terms use the rules of the game, i.e., they envision players, etc. But ultimately it is just the configuration itself that is one of these three.) In particular, o-winnable requires only that there *exists* a move for o that makes the configuration o-winnable. It is not necessary that "the game be allowed to continue, with o actually making such a move". Nor is it necessary that o be an expert player, make good moves, or indeed even be present. If you like, these terms refer to potentialities, not actualities.

2. Most people (including whoever it was who wrote the solution sheet) forgot that "0" is also a legal integer, and so must be encoded into something. Thus, for example, both solutions on the solution sheet are wrong. (The first could be correct by the change "... if it is negative *or zero.*"; the second by the change "... original integer was negative *or zero.*") (Recall that zero is neither positive nor negative.) This minor issue was assigned one point. All that counts about an integer is its *numerical value*, not the details of how it is written. Thus, one can write the integer "47" as "047" or "0047" or "+47" or "47.000" or "47" written with orange ink, but they all count as the same integer. In particular, then, one cannot encode additional information into the integer by written it in one of these alternative forms. The key thing about encoding is not only that it can be carried out (in this example, to obtain a positive integer from any given integer), but that it can also be decoded (in the example, to recover the given integer from the positive integer). Thus, for example, "Just drop the sign." and "Square the integer." are not legitimate encodings, for neither can be decoded. Thus, for the first, if I give you the result is "7" you do not know whether the original integer was "7" or "-7"; and similarly for the second. Here you were, of course, to encode a *single* integer into a positive integer (as opposed to several integers, before in class). (As you study this material, try to focus on what is really going on ("the idea of encoding") rather than the mechanics ("how to encode").) The class as a whole did rather well on this problem (which I thought would be more difficult). There were some very original encodings.

3. Please *explain your answer* briefly. There were a fair number of papers with "Just say 'terminates'." and no further explanation (these assigned 10 out of 16 points). It was also common to figure out, by some means, that every initial string would terminate, but then forget to state what the procedure is ("Just say 'terminates'.") (I assumed that people who had figured all this out knew also what the procedure is.) While most explanations of why every string must lead to termination were quite clear, a number were less so. What of

the answer “Let the technician play out the tag time, starting from the given initial string, and see what happens.”? Of course, this is general not a procedure, for it is conceivable that the technician could go on playing the game forever, without ever being able to announce “This game goes on forever”. It is curious, however, that *for particular example of the problem*, this actually is a procedure. The reason is that it is in fact the case that every game will terminate, and so the labors of the technician will also come to an end, with that technician announcing “terminates”. This is the old story: It need only be the case that the technician’s efforts terminate, not that the technician is aware of this. (I generally assumed that those giving this answer did not understand this point, unless there was some indication that they did.)

4. First, the structure of these questions was intended to be “We want to know, for each part, whether there is a procedure that, for every choice of the positive integer n, answers the question.” and not “The integer n is fixed at the beginning. Now we want to know, for each part, whether there is a procedure that, for that n, answers the question.” In short, the procedure is to be given before the n. Each part represents an infinite number of questions (one for each n), not a single question. I think that this was fairly clear from both wording and context, although the wording could perhaps have been improved.

a. The number π is *not exactly* 22/7. Indeed, π to six digits after the decimal is 3.141592, while 22/7 is 3.142857. The latter is only a convenient approximation to the former. “Irrational” means only “cannot be written as the quotient of two integers”, not “lacking rationality” or “having random digits”. Thus, for example, the number .121122111222111122222. . . is irrational. (Challenge for doubters: Find two integers such that one divided by the other gives this number.) Neither of these points is germane to the subject matter of this course, and neither had much impact on scoring, but I thought I would mention them. Very few had any real trouble with this part.

b. There was a tendency, here again, to lapse into an answer with no explanation (e.g., merely “We do not know”, which was assigned 5 points out of 10). Our doubts about their being a procedure arise not from the fact that there are an infinite number of questions (one for each n). Consider, for instance "Given positive integer n, is it prime?" This is an infinite number of questions (one for each n), and yet there certainly is a procedure for it. The doubts arise, rather, because even after we are given the n there is no sure-fire way of telling whether a "4" occurs after the nth digit of π. What of the answer "There is no procedure."? The fact that we may not happen to have a procedure in hand does not mean that none exists, only that we do not yet know. (This problem is a particularly good example of this point, for it turns out that there actually *does exist* a procedure for this particular question. But it is a subtle business.) We have as yet found no example in which we can guarantee that no procedure exists. One might even, by this point, begin to suspect that a procedure will always exist – there only remains to find it. *The entire thrust of this course is to produce an example of a question* (actually, an infinite number) *for which we can show that no procedure exists.* (Those who said "no procedure" for both parts b and c, for the same reason, were not penalized twice.)

c. It was intended that the "encoding" for this part be "our particular encoding", but it does not seem to make any real difference. The whole idea of "either… or…" was apparently confusing, and I do not think that the note on the board helped much. The question "Is n either… or…" is to have the same meaning as "Is the dog either brown or wet?" The answer is "yes" if the dog is either brown or wet, or both, and "no" otherwise. So, each n either has the property in question (either.… or…), and so for each n the answer is either "yes" or "no". But there are an infinite number of questions, namely one for each n. In particular, you are not being asked, for each n, to say which is the case (just as for the dog you are not being asked whether brownness or wetness is the case). You are not being asked for separate procedures for encoding and for primes. The issue, rather, is a procedure for the entire question. There is no simple general slogan for deciding whether there is a procedure for "or" or "and"-questions: One just has to think it through each time. (By the way, the integer "1" is not, by definition, a prime.)

The class as a whole did very well on this test. Your answers were generally clear, brief, and with good explanations. The next test will be rather like this one, but I will try to make it a little harder (not too much!), and a little more interesting. If you have any suggestion about testing in this course, I would very much appreciate them. If you feel your paper was misgraded, please follow the procedure given earlier.

Phy Sci 113 Nov 22, 1988

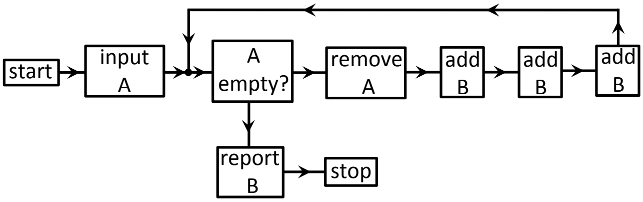
Test 2

Name:

Note: "Draw flow chart" means using only our seven instructions, with no "dashed boxes".

1. (12 points) Draw a flow chart that accepts as input nonnegative integer n; and stops if n is odd and continues forever if n is even.

2. (14 points) What does the flow chart below do? What would it do if the instruction "remove A" were replaced by "add A"?



3. (10 points) Draw the flow chart represented by the following sequence of integers: 1, 3, 1, 2, 4, 1, 3, 2.

4. (33 points) Answer ("yes", "no", or "we do not know"), and

explain your answer *briefly.*

a. Does there exist a flow chart with no inputs that reports "0" if there exist no positive integers x, y, z, and n ≥ 3 with $x^n + y^n = z^n$; and "1" if there do exist such integers?

b. Does there exist a flow chart that accepts as input a single integer; and reports whether or not that integer is the encoding (by our encoding scheme) of some flow chart?

c. Does there exist a flow chart that accepts as input an integer that encodes some flow chart having no inputs; and reports whether or not that flow chart, when executed, will stop?

5. (24 points) Definition: A number is said to be *computable* if there exists a flow chart that accepts as input positive integer n; and reports the nth digit in the decimal expansion of that number.

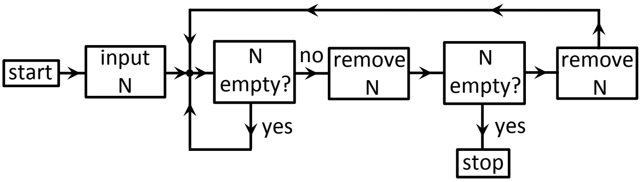
a. Draw the flow chart that shows that 1/4 is computable.

b. Give any example of a number that is not computable.

Phy Sci 113 Nov 22, 1988

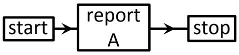
Test 2 — Solutions

1.

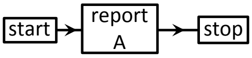


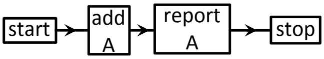
2. The flow chart accepts as input a nonnegative integer; and reports three times that integer. As modified, the flow chart would accept as input a nonnegative integer; and report “0” and stop if that integer is zero, and continue forever if that integer is other than zero.

3.



4. a. There does exist such a flow chart. Indeed, this is a single “yes-or-no” question, and so the desired flow chart is one of these:

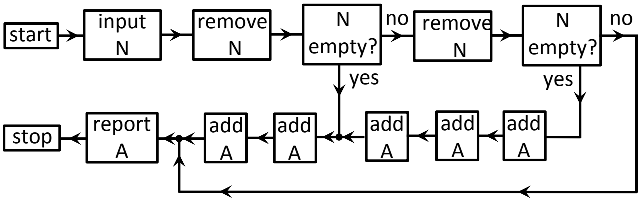




b. There does exist such a flow chart. There certainly exists a procedure to decide whether or not an integer is the encoding of some flow chart. But we have argued that every procedure can be rendered by a flow chart.

c. There does not exist such a flow chart. Indeed, assuming there existed such a flow chart, we could construct a flow chart Nancy, as follows. Let Nancy i) take the flow chart encoded at X, ii) replace its instruction “input A” by Y copies of “add A”, iii) encode the resulting flow chart (which now has no input instructions) as an integer, and finally iv) run the flow chart whose existence we assumed above. Note that this would indeed be a Nancy. But our theorem says there exists no Nancy. Hence, there cannot exist a flow chart of the type described.

5. a. The flow chart must report "2" when "1" is input, "5" when "2" is input, and "0" otherwise.



b. Consider the number whose n^{th} digit is "1" if n is the encoding of a flow chart (of our special form) and A-value such that that flow chart, with that A-value, stops; and "2" otherwise. Were this number computable, then, clearly, one could construct a flow chart Nancy. But there exists no Nancy, and so this number cannot be computable.

Phy Sci 113 Nov 29, 1988

Test 2 — Comments

This test was harder than the first one. Indeed, the average score, 52, was lower than that of Test 1, although the maximum number of points, 93 was higher. Here is the distribution of scores:

Score	Nmbr receiving	Score	Numbr receiving
10-19	1	50-59	55
20-29	5	60-69	27
30-39	16	70-79	10
40-49	39	80-89	1

Comments on the individual problems:

1. This problem was basically routine, the little twist being that one had to have one's flow chart, not merely *report* the answer, but rather *do* something in response to that answer. People did well on it on the whole, with mostly minor, technical errors. For example, one is not supposed to have two flow lines entering an instruction box. (It is clear what is meant. Logically, but we have adopted the convention of one line entering a box.) One must use, instead, a node.

2. What a flow chart *does* refers to whether it stops or not, and its communications with us (i.e., its inputs and reports). Thus, for the first part of the problem, one should respond "Reports three times the integer input.", rather than "Computes three times A and stores it in B." (The latter was assigned five points out of the seven points allocated to this part of the problem.) Similarly, for the second part, what counts is that, for A greater than zero, the execution never stops, rather than that it causes bowls A and B to contain more and more beans. With these flow charts, one has to be a little careful to check the special cases, e.g., A = 0 here, and N = 0 in problem 1. It is not uncommon that a flow chart does the correct thing "in general", but fails to do so for such a special case. The class did well on this problem.

3. Virtually everyone got this easy problem. A few had not, apparently, written down on their crib sheets the rules for encoding flow

charts, and there were a couple of people who felt compelled to incorporate an "input A" somewhere in the flow chart.

4. a. This is a *single* question, with answer either "yes" or "no". As such, there *must* exist a flow chart to answer it. This is the same old story we saw before, for procedures. By far the most common answer was "One can keep searching for such x, y, z, n; but cannot be sure of finding any. So, we do not know whether or not there is a flow chart." (This was assigned 3 points out of the 11 for this part.) Thus, in this sense, too, flow charts are identical to procedures. A few people had trouble with the idea of a flow chart with no "input" instruction. This is fine, as long as the flow chart does what it is suppose to do. It is just that. Lacking an input, a flow chart can only "do" one thing, as opposed to a variety of things.

b. Here, 4 points were assigned to the answer ("yes"), the remaining 11 to the reason for that answer. It was surprisingly common for people to explain why there is a procedure for deciding whether an integer is the encoding of *other integers*, without any direct reference to whether it is the encoding of a flow chart. (This response was assigned 4 (for answer) + 2 (for reason) = 6.) Note that not every sequence of integers is the encoding of some flow chart, and, indeed, the conditions for such a sequence to be such an encoding are quite stringent. (One could, in principle, write them all out, but the result would be very long.) One should not think of there as being "two different kinds" of integers – real, ordinary integers and integers that are encodings of flow charts. An integer is an integer: It is just that one can interpret integers in various ways.

c. This was a hard problem (as promised): I think only one person saw it all the way through. The ides is that Nancy deals only with flow charts with single inputs, whereas here we are asking about flow charts with no inputs. But if one did have the flow chart asked for here, then one could use it to build a Nancy. The reason is that one can always suppress the "input" in the flow charts that Nancy examines, replacing it with a suitable number of "add" instructions. Anyway, this is a tricky point – and a good example of the kinds of arguments one makes using the "There exists no Nancy."-theorem. One shows that procedures for other things do not exist, for it if they did then one could construct a Nancy. I assigned 4 points for the answer ("no"), no matter what else was said. A few thought that the desired flow chart would have to be *exactly* a Nancy (not noticing the difference in inputs). There were quite a few papers with an intuitive sense of the answer, arguing that the desired flow chart would act much "like Nancy" (despite the difference in inputs), and so was unlikely to exist (these assigned 7 out of maximum 11). If you wish to understand how our theorem really works, understand the answer to this problem.

5. This problem, unfortunately, floundered because the definition turned out to be confusing. Apparently, there is a special style to mathematical definitions: they are hard to get if one is not used to it. It seemed, with all the explanations during the test, most people ultimately understood, basically, what the definition said. (In fact, the notion of a computable number is an important one in mathematics.)

a. Many of the errors, unfortunately, were traceable to problems with the basic definition. Thus, some wanted to "input" 1/4 at A, while others wanted to input the digits of 1/4; 2, 5, 0,... But one is supposed to input n – which digit one wants – and have the flow chart report the digit. Thus, if 1 is input, 2 is to be reported; if 2 input, 5 reported; if 3 input, 0 reported; etc. The "later" digits of .25, by the way, are "0" 's: 1/4 = .25000000000... A few who asked during the test were told that one counts the digits from the left (for there is no way to do it from the right) – but I do not think this was a real problem for anyone.

b. This was the second hard problem. It gave rise to an additional (unintended) issue: What, exactly, is a "number"? For our purposes, one can think of a number as being nothing more and nothing less than its digits. Thus, to "specify" a number, one need only give the *rule* for determining its digits, in succession. (Note here "rule" and not "procedure".) Thus, for example, "1/0" and "infinity" are not numbers, for they give rise to no rules. (A few people wanted to go to imaginary numbers. They won't work because, if they are to be dealt with by flow charts, one will have to encode them to integers. But, once so encoded, they lose their apparent "noncomputability".) What the question asks, then, is for a number (i.e., a rule for the digits), without a procedure for the digits. To find one, then, one must find a "question" (which of course, will determine the digits) that has an answer, but no procedure for that answer. But we have just such a thing: the question of whether flow charts stop. So, we obtain a noncomputable number by making its digits depend on what happens to flow charts. In any case, it is very tricky, and I did not expect many to get it. Note, for example, that pi is very computable (and, indeed, we have even discussed briefly, the procedure for computing its digits). In any case, I did not expect people to find this part easy – and you didn't. A few, however, did have a general idea of how to proceed.

In terms of difficulty, etc. the final will be rather like this test.

Phy Sci 113 Dec 6, 1988

Final Examination

Name:
Please answer and *briefly explain* your answer. “Does there exist flow chart?” may require answer “We don’t know”.

1. (20 points) consider the tic-tac-toe configuration at the right.

a. Suppose first that it is X’s move. Of the five possible moves for X, which are good moves?

X	O	O
	X	

b. Now suppose instead that it is O’s move. Is this configuration X-winnable, O-winnable, or drawable?

2. (14 points) Draw a flow chart (using only the 7 basic instructions!) that accepts integer n, and reports whether or not that integer is greater than 4.

3. (26 points) Consider the following tag game: The alphabet consists of “a”, “b”, and “c”; for “a” leftmost, “bc” is appended on the right, for “b”, “acb”, and for “c”, “bac”; and in every case the leftmost three letters are deleted. We wish to construct a flow chart

that will accept as input any initial string (suitably encoded), and report whether the resulting tag game will terminate, cycle, or neither.

a. describe any suitable encoding for the initial strings.

b. Does there exist such a flow chart?

4. (20 points) We wish to construct a flow chart that will decide whether or not any given candidate for a Nancy in fact works as a Nancy.

a. Now many inputs would such a flow chart have? (explain briefly.)

b. Does there exist such a flow chart?

5. (30 points) For each of the questions below, we wish to construct a flow chart to answer the question(s). How many inputs would such a flow chart have? Does there exist such a flow chart?

a. Given integer n from 1 to 9, do there exist 100 consecutive n's in the decimal expansion of π?

b. Is it the case that every integer is the difference of two primes?

c. Given integer n, do there exist positive integers x, y, and z with $x^n + y^n = z^n$?

6. (14 points) In our proof of the nonexistence of a flow chart Nancy, we showed that, given any candidate for a Nancy, there exists one particular set of X- and Y-values for which this candidate gives the wrong answer. Show that there exists at least one other such set of values.

7. (26 points) Does there exist a flow chart that accepts, at “input X”, any number that encodes a flow chart of our special form, and, at “input Y”, any nonnegative integer, and

a. Stops if that flow chart with that A-value stops, and continues forever if it continues forever?

b. Stops if that flow chart with that A-value continues forever, and continues forever if it stops?

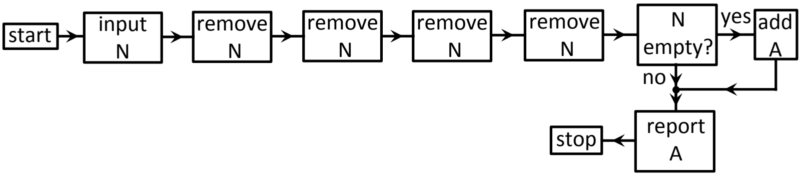
Phy Sci 113 Dec 6, 1988

Final Exam — Solutions

1. a. The move at the middle of the bottom row is not a good move. All others are. This is an X-winnable configuration, and so a move, to be a good move, need only result in an X-winnable configuration. This single move does not; and the other four do.

b. This configuration is drawable, That is, it is possible for O, against any play by X, to achieve at least a draw.

2.



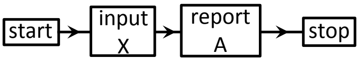
Here, "0" is reported if N is greater than four, and "1" otherwise.

3. a. We could, for example, encode an initial string by the integer having one digit for each letter in that string, where "1" stands for "a", "2" for "b", and "3" for "c". The empty string would be represented as "0". Thus, for example, initial string "bbac" would be encoded as integer "2213".

b. There does exist such a flow chart. First note that, since in no case are more than three letters appended on the right while three letters are always deleted on the left, the initial string can never grow in size. Hence, every string will either terminate or cycle: The game cannot go on forever without doing one of these two. But these two possibilities could be recognized by a flow chart. Hence, we need merely construct a flow chart that plays out the tag game, checking after each step to see whether the tag game has terminated or cycled. Eventually, one will occur.

4. a. The flow chart would have a single input. Into this input would be entered the encoding of our given candidate for Nancy.

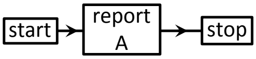
b. There does exist such a flow chart. Indeed, here it is:



Here, our flow chart reports "0" if the candidate (encoded at X) is not a Nancy, and "1" if it is. (Recall: There exists no Nancy!)

5.a. The flow chart would have a single input, into which we are to enter the integer n (from 1 to 9). There does exist such a flow chart. There are only nine questions total, each of which has answer "yes" or "no". Hence, there is a grand total of $2^9 = 512$ possible sets of answers. For each of these, we can draw a flow chart. One (although we do not know which one!) is the correct flow chart.

b. This is a single "yes-or-no" question. There would be no inputs. There does exist such a flow chart. Indeed, here it is:



Here, "0" is reported for "no", and "1" for "yes". (Recall: The integer "19" is not the difference of two primes.)

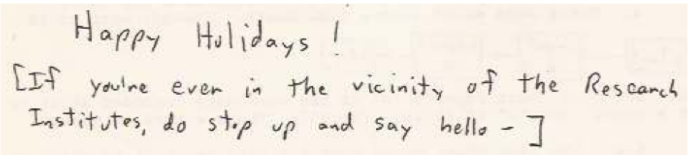
c. There would be a single input, into which would be entered the value of n. We do not know whether or not there exists such a flow chart.

6. Fix a candidate for flow chart Nancy. The X- and Y-values we used in our proof were x= G and Y = G, where G is the integer that encodes a certain flow chart George constructed from our candidate for Nancy. Let us now construct a new flow chart, call it Henry, that is exactly the same as George, except that an extra "report S" is added just before the instruction "stop". (Almost any non-substantive change in George would do here.) Now try our candidate for Nancy again, but which X = H and Y = H, where H is the integer that encodes this flow chart Henry. But Nancy must also give the wrong answer for these X- and Y-values, by exactly the same argument as for George. Here, then, is a second such set of values. (Note that H must

be a different integer from G, since Henry is a different flow chart from George.)

7. a. There does exist such a flow chart. As we have discussed, there exists a flow chart that merely "runs the flow chart encoded at X, with A-value at Y, doing whatever it would do". Such a flow chart would do nicely here.

b. There does not exist such a flow chart. Suppose, for a moment, that we had one. Then we could construct a flow chart Nancy. We would build a flow chart that first runs the flow chart answering (a) for ten steps, then the flow chart answering (b) for ten steps, then the one answering (a) for ten steps then the one answering (b) . . . Eventually, either the flow chart answering (a) or the flow chart answering (b) would have to stop. By determining which one stops, and reporting the result (as "stops" if the flow chart answering (a) stops, and "continues forever" if the flow chart answering (b) stops), we would have our Nancy. But there does not exist a flow chart Nancy. Hence, there cannot exist the flow chart required for part (b).



Happy Holidays !

[If you're ever in the vicinity of the Research Institutes, do stop up and say hello –]

About the author

Robert Geroch is a theoretical physicist and professor at the University of Chicago. He obtained his Ph.D. degree from Princeton University in 1967 under the supervision of John Archibald Wheeler. His main research interests lie in mathematical physics and general relativity.

Geroch's approach to teaching theoretical physics masterfully intertwines the explanations of physical phenomena and the mathematical structures used for their description in such a way that both reinforce each other to facilitate the understanding of even the most abstract and subtle issues. He has been also investing great effort in teaching physics and mathematical physics to non-science students.

Robert Geroch with his dog Rusty