EXCERPTED FROM

STEPHEN WOLFRAM

A NEW KIND OF SCIENCE

SECTION 11.6

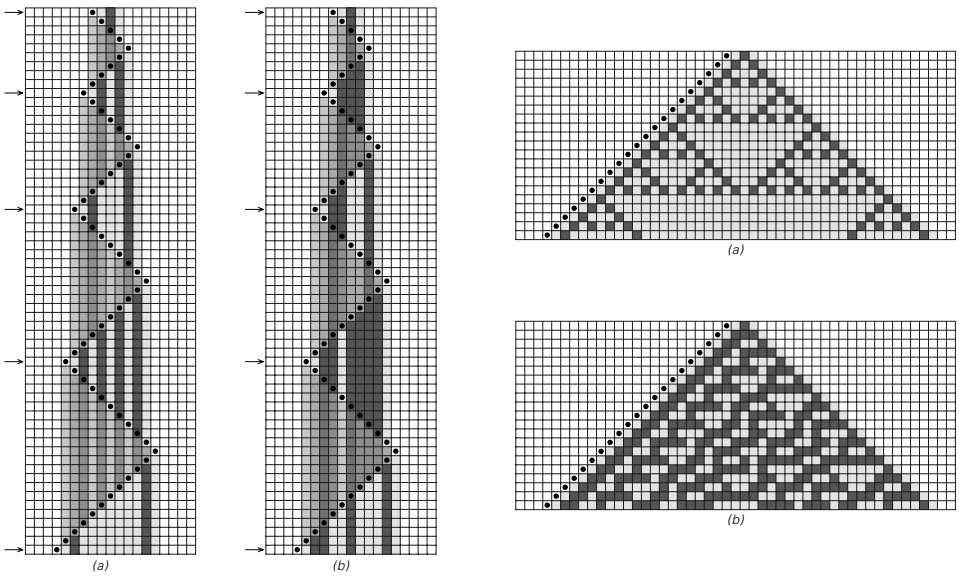
Emulating Cellular Automata with Other Systems

Emulating Cellular Automata with Other Systems

In the previous section we discovered the rather remarkable fact that cellular automata can be set up to emulate an extremely wide range of other types of systems. But is this somehow a special feature of cellular automata, or do other systems also have similar capabilities?

In this section we will discover that in fact almost all of the systems that we considered in the previous section—and in Chapter 3—have the same capabilities. And indeed just as we showed that each of these various systems could be emulated by cellular automata, so now we will show that these systems can emulate cellular automata.

As a first example, the pictures below show how mobile automata can be set up to emulate cellular automata. The basic idea is to have the active cell in the mobile automaton sweep backwards and forwards, updating cells as it goes, in such a way that after each complete sweep it has effectively performed one step of cellular automaton evolution.



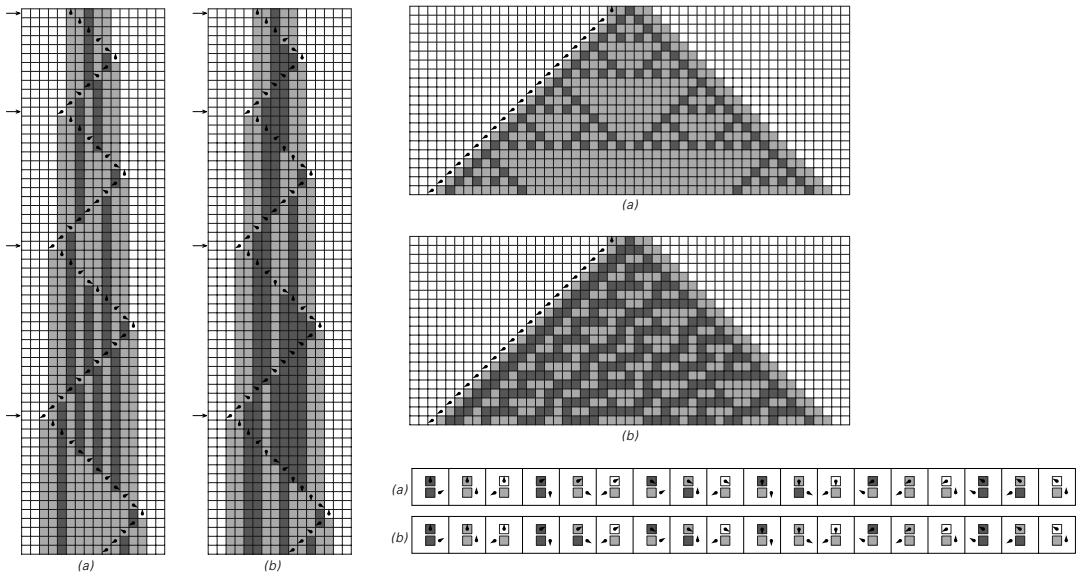
Examples of mobile automata emulating cellular automata. In case (a) the rules for the mobile automaton are set up to emulate the rule 90 elementary cellular automaton; in case (b) they are set up to emulate rule 30. The pictures on the right are obtained by keeping only the steps indicated by arrows on the left, corresponding to times when the active cell in the mobile automaton is further to the left than it has ever been before. The mobile automata used here involve 7 possible colors for each cell.

The specific pictures at the bottom of the facing page are for elementary cellular automata with two possible colors for each cell and nearest-neighbor rules. But the same basic idea can be used for cellular automata with rules of any kind. And this implies that it is possible to construct for example a mobile automaton which emulates the universal cellular automata that we discussed a couple of sections ago.

Such a mobile automaton must then itself be universal, since the universal cellular automaton that it emulates can in turn emulate a wide range of other systems, including all possible mobile automata.

A similar scheme to the one for mobile automata can also be used for Turing machines, as illustrated in the pictures below. And once again, by emulating the universal cellular automaton, it is then possible to construct a universal Turing machine.

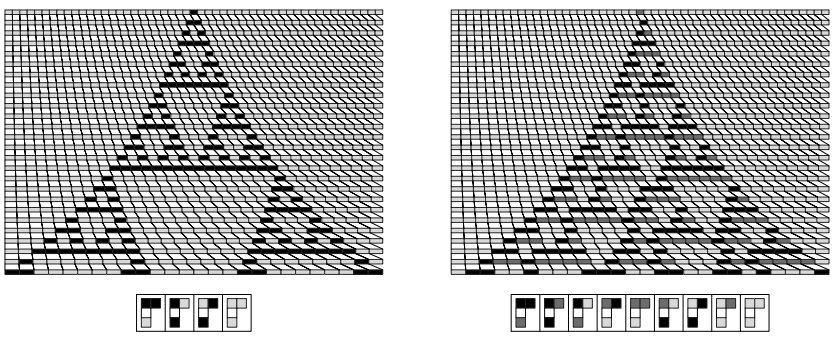
But as it turns out, a universal Turing machine was already constructed in 1936, using somewhat different methods. And in fact that universal Turing machine provided what was historically the very first clear example of universality seen in any system.



Examples of Turing machines that emulate cellular automata with rules 90 and 30. The pictures on the right are obtained by keeping only the steps indicated by arrows on the left. The Turing machines have 6 states and 3 possible colors for each cell.

Continuing with the types of systems from the previous section, we come next to substitution systems. And here, for once, we find that at least at first we cannot in general emulate cellular automata. For as we discussed on page 83, neighbor-independent substitution systems can generate only patterns that are either repetitive or nested—so they can never yield the more complicated patterns that are, for example, needed to emulate rule 30.

But if one generalizes to neighbor-dependent substitution systems then it immediately becomes very straightforward to emulate cellular automata, as in the pictures below.

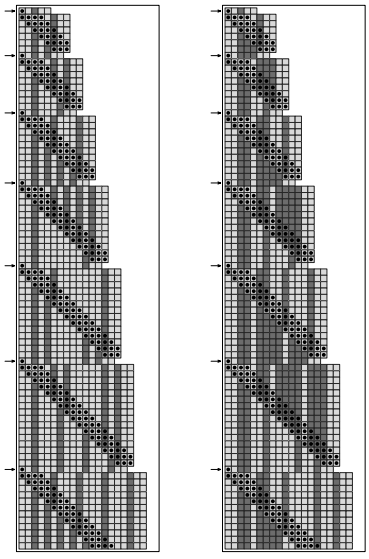


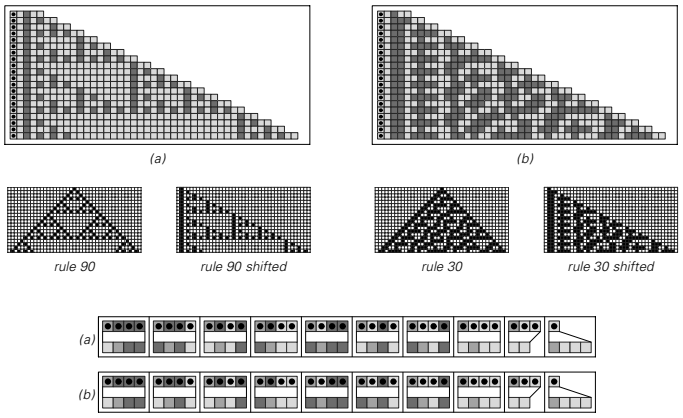
Neighbor-dependent substitution systems that emulate cellular automata with rules 90 and 30. The systems shown are simple examples of neighbor-dependent substitution systems with highly uniform rules always yielding just one cell and corresponding quite directly to cellular automata.

What about sequential substitution systems? Here again it turns out to be fairly easy to emulate cellular automata—as the pictures at the top of the facing page demonstrate.

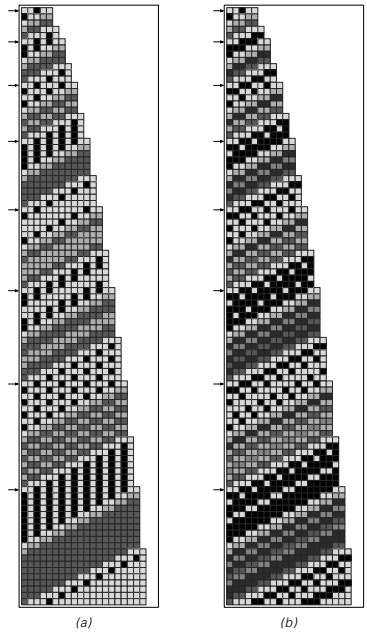
Perhaps more surprisingly, the same is also true for ordinary tag systems. And even though such systems operate in an extremely simple underlying way, the pictures at the bottom of the facing page demonstrate that they can still quite easily emulate cellular automata.

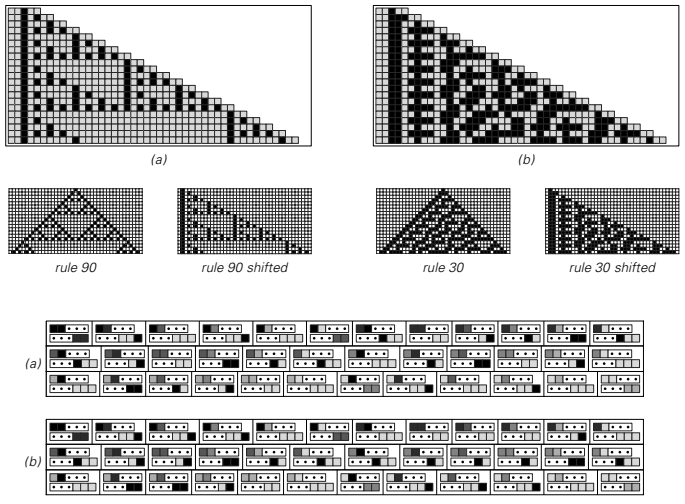
What about symbolic systems? The structure of these systems is certainly vastly different from cellular automata. But once again—as the picture at the top of page 668 shows—it is quite easy to get these systems to emulate cellular automata.



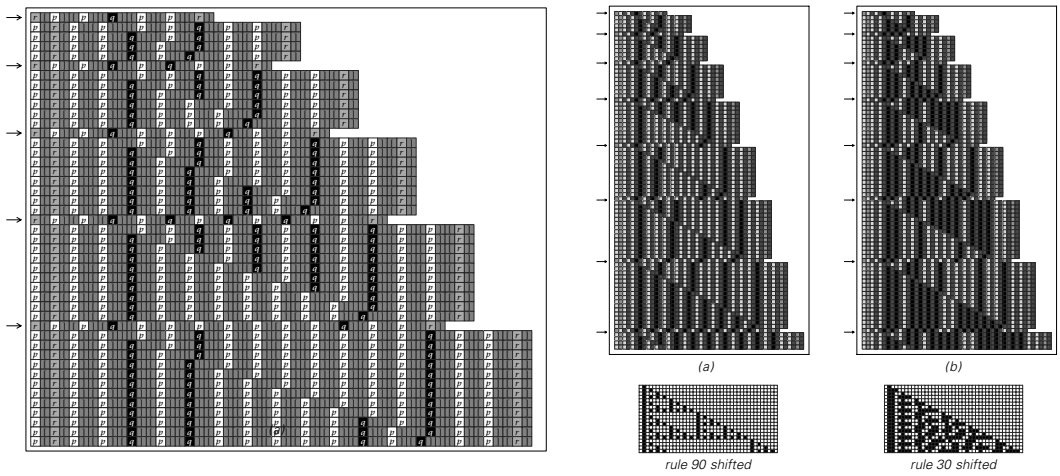


Sequential substitution systems that emulate cellular automata with rules 90 and 30. The pictures at the top above are obtained by keeping only the steps indicated by arrows on the left. The sequential substitution systems involve elements with 3 possible colors.





Tag systems that emulate the rule 90 and rule 30 cellular automata. The pictures at the top above are obtained by keeping only the steps indicated by arrows on the left. Both tag systems involve 6 colors.



(a) $p[x\_][p][p][p] \to p[x[p]][p][p],\ p[x\_][p][p][q] \to p[x[q]][p][q],\ p[x\_][p][q][p] \to p[x[p]][q][p],\ p[x\_][p][q][q] \to p[x[q]][q][q],\ p[x\_][q][p][p] \to p[x[q]][p][p],$
$p[x\_][q][p][q] \to p[x[p]][p][q],\ p[x\_][q][q][p] \to p[x[q]][q][p],\ p[x\_][q][q][q] \to p[x[p]][q][q],\ r[x\_] \to p[r[p][p]][x],\ p[x\_][p][p][r] \to x[p][p][r]$

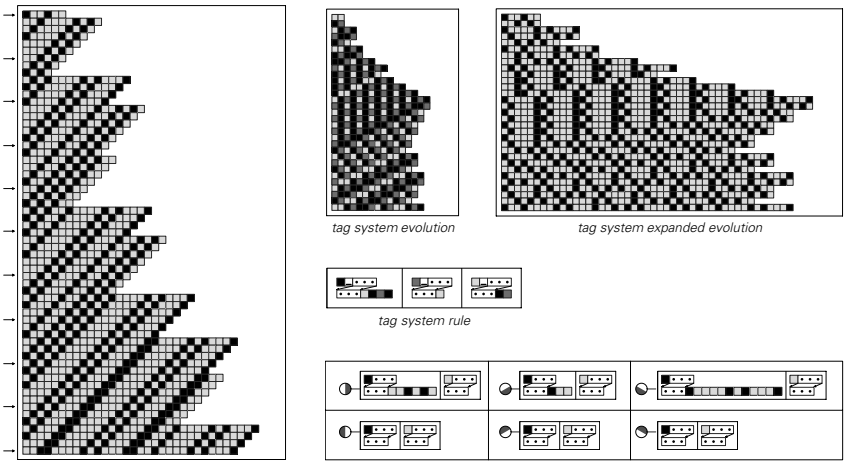
(b) $p[x\_][p][p][p] \to p[x[p]][p][p],\ p[x\_][p][p][q] \to p[x[q]][p][q],\ p[x\_][p][q][p] \to p[x[q]][q][p],\ p[x\_][p][q][q] \to p[x[q]][q][q],\ p[x\_][q][p][p] \to p[x[q]][p][p],$
$p[x\_][q][p][q] \to p[x[p]][p][q],\ p[x\_][q][q][p] \to p[x[p]][q][p],\ p[x\_][q][q][q] \to p[x[p]][q][q],\ r[x\_] \to p[r[p][p]][x],\ p[x\_][p][p][r] \to x[p][p][r]$

Symbolic systems set up to emulate cellular automata that have rules 90 and 30. Unlike the examples of symbolic systems in Chapter 3, which involve only one symbol, these symbolic systems involve three symbols, p, q and r.

And as soon as one knows that any particular type of system is capable of emulating any cellular automaton, it immediately follows that there must be examples of that type of system that are universal.

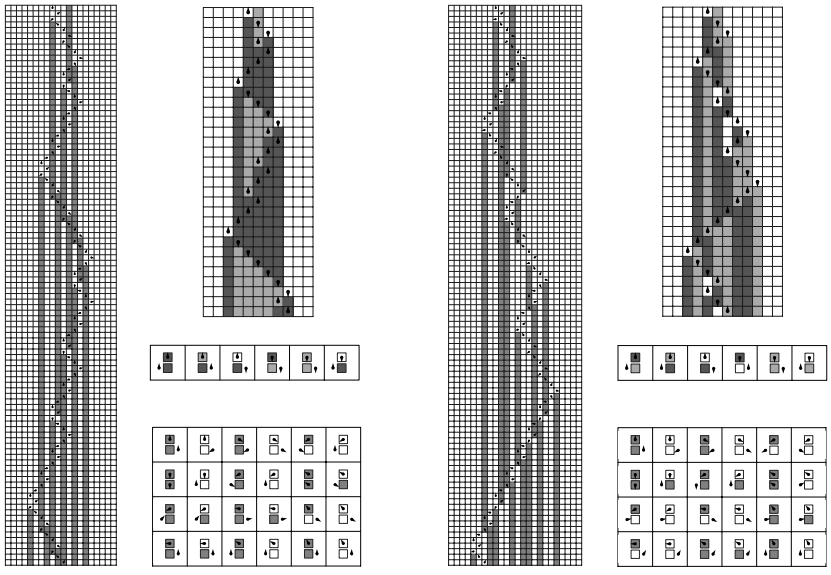
So what about the other types of systems that we considered in Chapter 3? One that we have not yet discussed here are cyclic tag systems. And as it turns out, we will end up using just such systems later in this chapter as part of establishing a dramatic example of universality.

But to demonstrate that cyclic tag systems can manage to emulate cellular automata is not quite as straightforward as to do this for the various kinds of systems we have discussed so far. And indeed we will end up doing it in several stages. The first stage, illustrated in the picture at the top of the facing page, is to get a cyclic tag system to emulate an ordinary tag system with the property that its rules depend only on the very first element that appears at each step.

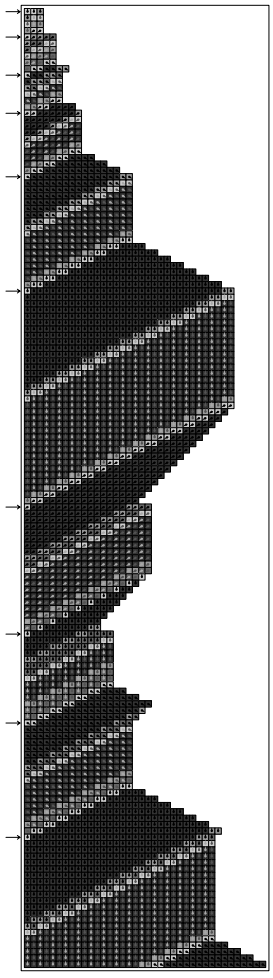


A cyclic tag system emulating a tag system that depends only on the first element at each step. In the expanded tag system evolution, successive colors of elements are encoded by having a black cell at successive positions inside a fixed block of white cells.

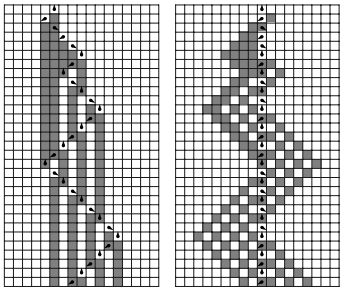
And having done this, the next stage is to get such a tag system to emulate a Turing machine. The pictures on the next page illustrate how this can be done. But at least with the particular construction shown, the resulting Turing machine can only have cells with two possible colors. The pictures below demonstrate, however, that such a Turing



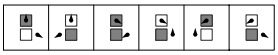
Turing machines with two colors emulating ones with more colors.



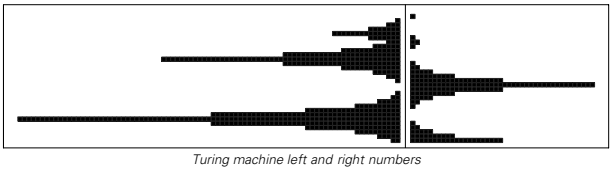
tag system evolution (150 steps)



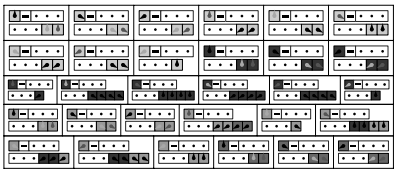
Turing machine evolution



Turing machine rule



Turing machine left and right numbers



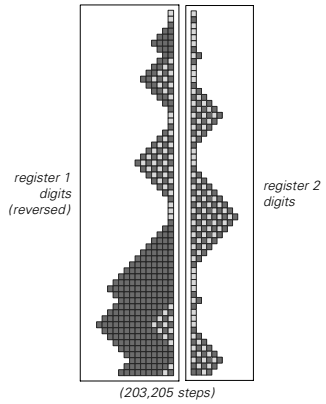
tag system rule

tag system compressed evolution (1500 steps)

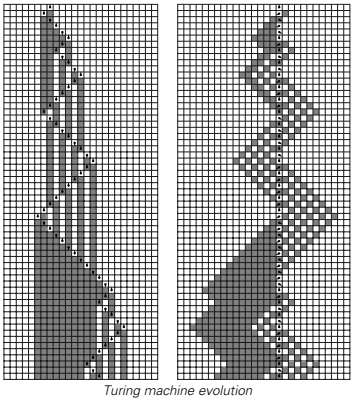
Emulating a Turing machine with a tag system that depends only on the first element at each step. The configuration of cells on each side of the head in the Turing machine is treated as a base 2 number. At the steps indicated by arrows the tag system yields sequences of dark cells with lengths that correspond to each of these numbers.

machine can readily be made to emulate a Turing machine with any number of colors. And through the construction of page 665 this then finally shows that a cyclic tag system can successfully emulate any cellular automaton—and can thus be universal.

This leaves only one remaining type of system from Chapter 3: register machines. And although it is again slightly complicated, the pictures on the next page—and below—show how even these systems can be made to emulate Turing machines and thus cellular automata.

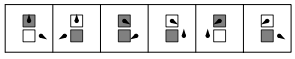


(203,205 steps)

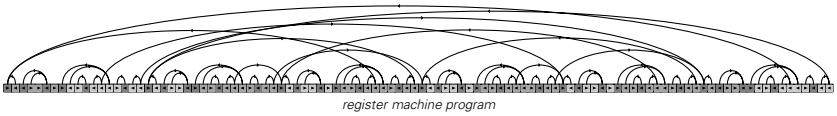


Turing machine evolution

A register machine emulating a slightly more complicated Turing machine than on the next page.

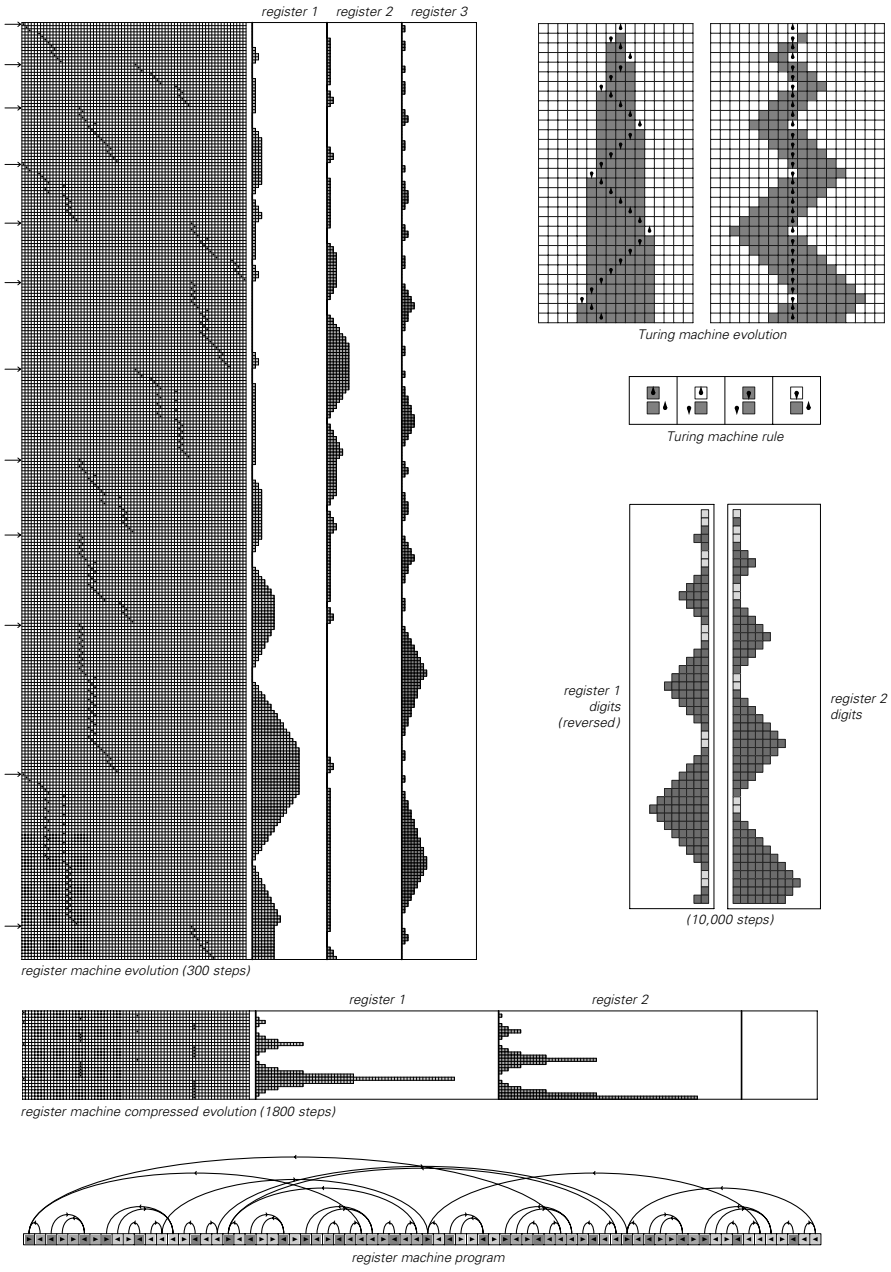


Turing machine rule



register machine program

So what about systems based on numbers, like those we discussed in Chapter 4? As an example, one can consider a generalization of the arithmetic systems discussed on page 122—in which one has a whole number n, and at each step one finds the remainder after dividing by a constant, and based on the value of this remainder one then applies some specified arithmetic operation to n.

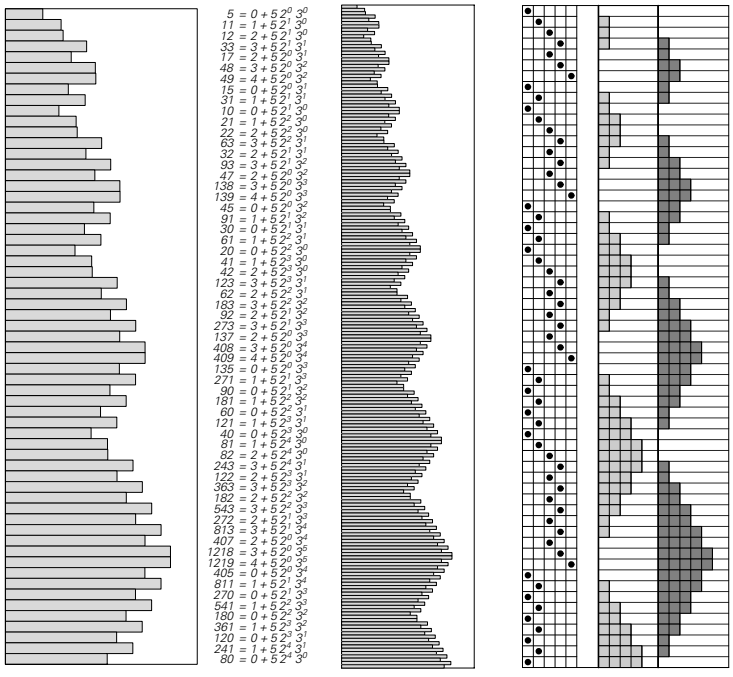


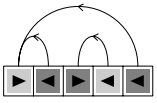
An example of a register machine set up to emulate a Turing machine. The Turing machine used here has two states for the head; the register machine program has 72 instructions and uses three registers. The register machine compressed evolution keeps only steps corresponding to every other time the third register gets incremented from zero.

The picture below shows that such a system can be set up to emulate a register machine. And from the fact that register machines are universal it follows that so too are such arithmetic systems.

And indeed the fact that it is possible to set up a universal system using essentially just the operations of ordinary arithmetic is closely related to the proof of Gödel's Theorem discussed on page 784.

But from what we have learned in this chapter, it no longer seems surprising that arithmetic should be capable of achieving universality. Indeed, considering all the kinds of systems that we have found can exhibit universality, it would have been quite peculiar if arithmetic had somehow not been able to support it.





An example of how a simple arithmetic system can emulate a register machine. The arithmetic system takes the value n that it obtains at each step, computes $Mod[n, 30]$, and then depending on the result applies to n one of the arithmetic operations specified by the rule on the left below. The rule is set up so that if the value of n is written in the form $i + 5$, 2^a, 3^b then the values of i, a and b on successive steps correspond respectively to the position of the register machine in its program, and to the values of the two registers (2 and 3 appear because they are the first two primes; 5 appears because it is the length of the register machine program). The values of n in the pictures on the left are indicated on a logarithmic scale.

$2n+1$	$(n-1)/3$	$3(n-1)$	$(n+1)/2$	$(n-4)/3$	$2n+1$	$n+1$	$3(n-1)$	$n+1$	$n+1$
0	1	2	3	4	5	6	7	8	9
$2n+1$	$n+1$	$3(n-1)$	$(n+1)/2$	$n+1$	$2n+1$	$(n-1)/3$	$3(n-1)$	$n+1$	$(n-4)/3$
10	11	12	13	14	15	16	17	18	19
$2n+1$	$n+1$	$3(n-1)$	$(n+1)/2$	$n+1$	$2n+1$	$n+1$	$3(n-1)$	$n+1$	$n+1$
20	21	22	23	24	25	26	27	28	29