Infinite versions of minesweeper are Turing complete

Richard Kaye
School of Mathematics
The University of Birmingham
Birmingham
B15 2TT

R.W.Kaye@bham.ac.uk
http://www.mat.bham.ac.uk/R.W.Kaye

31st May 2007

The starting point for this paper is my article [5] showing that the well-known *Minesweeper* game is NP-complete. The proof was by making suitable minesweeper configurations simulate digital computers, with logic gates such as AND and NOT gates. This is reminiscent of John Conway's game of life [2] which was proved to be Turing complete by similar means [1]. (For background information on Turing machines and computability please see just about any text book on these. I found the account by Wang [7] particularly enjoyable and can recommend it as it presents just about the right about of material needed here in a straightforward and not too technical manner.)

It is reasonable to ask whether or not there is an version of *Minesweeper* analogous to *Life* that is also Turing complete. In this paper I show that there is.

Before I start, we should consider what this question asks for. Games like *Life* and *Minesweeper* are played on a grid, with each space on the grid labelled with at most one of only finitely many symbols. Such games played on a finite grid will typically be computable—by laboriously checking all possible combinations—so we are interested in versions of *Minesweeper* played on an infinite grid. Both *Minesweeper* and *Life* are normally played on square grids in the plane with an adjacency relation between two neighbouring squares that includes diagonals, and this is what I shall consider here, though other grids may well be of interest. The main feature of *Minesweeper* is that knowledge of a symbol in one square gives partial information about the *number* of each kind of symbol in the eight neighbouring squares without providing any direct information about which squares contain what, and it is this feature of the minesweeper game I wish to preserve in the infinite versions I consider.

I propose to 'play' *Minesweeper* with an infinite square grid, each square being labelled by a symbol from a finite set Σ, or with a 'blank' (meaning no information is provided), and the rules being give by a table, of the form

	a	b	c	$\dots$	z
a	n_{aa}	n_{ab}	n_{ac}	$\dots$	n_{az}
b	n_{ba}	n_{bb}	n_{bc}	$\dots$	n_{bz}
c	n_{ca}	n_{cb}	n_{cc}	$\dots$	n_{cz}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
z	n_{za}	n_{zb}	n_{zc}	$\dots$	n_{zz}

where each n_{xy} is one of 0, 1, 2, 3, 4, 5, 6, 7, 8, or ?, this being the precise number of squares containing the symbol y from the 8 squares adjacent to a square containing a symbol x. The entry ? means we are not given any information. (I have taken $\Sigma = \{a, b, c, \dots, z\}$ here for the purposes of illustration only.)

Thus, for the table to be reasonable, we should expect that

$$0 \leq \sum_{y \in \Sigma} n_{xy} \leq 8$$

for each x, counting ? as zero here, with at least one ? in each row that does not add up to exactly 8.

For example, the rules of the usual minesweeper game are expressed with set of symbols $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, *\}$ and table

	0	1	2	3	4	5	6	7	8	$*$
0	?	?	?	?	?	?	?	?	?	0
1	?	?	?	?	?	?	?	?	?	1
2	?	?	?	?	?	?	?	?	?	2
3	?	?	?	?	?	?	?	?	?	3
4	?	?	?	?	?	?	?	?	?	4
5	?	?	?	?	?	?	?	?	?	5
6	?	?	?	?	?	?	?	?	?	6
7	?	?	?	?	?	?	?	?	?	7
8	?	?	?	?	?	?	?	?	?	8
$*$	?	?	?	?	?	?	?	?	?	?

The game of minesweeper with a given rule table is to determine the possible configurations; more often, given some partial configuration (usually finite) determine its possible continuations over the whole plane (if any!)

Example 1. *For the set $\Sigma = \{A, B\}$ and rules table*

	A	B
A	?	?
B	?	4

it seems to be an interesting problem to find all the possible configurations of the whole plane. For example, the following are possible,

$$\begin{array}{ccccccccc}
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \\
\cdots & A & B & A & B & A & B & \cdots \\
\cdots & B & A & B & A & B & A & \cdots \\
\cdots & A & B & A & B & A & B & \cdots \\
\cdots & B & A & B & A & B & A & \cdots \\
\cdots & A & B & A & B & A & B & \cdots \\
\cdots & B & A & B & A & B & A & \cdots \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & &
\end{array}
\qquad
\begin{array}{ccccccccc}
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \\
\cdots & A & B & A & A & B & A & \cdots \\
\cdots & B & B & B & B & B & B & \cdots \\
\cdots & A & B & A & A & B & A & \cdots \\
\cdots & A & B & A & A & B & A & \cdots \\
\cdots & B & B & B & B & B & B & \cdots \\
\cdots & A & B & A & A & B & A & \cdots \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & &
\end{array}$$

With questions like this in mind we can formulate

The general consistency problem: on input a rule table and a finite partial configuration, determine if there is a complete labelling of the whole plane that obeys the rules and extends the input configuration.

and the specific consistency problem for a given rule table R

The consistency problem for R: on input a finite partial configuration, determine if there is a complete labelling of the whole plane that obeys the rules in R and extends the input configuration.

I shall show here that

Theorem 2. *There is a specific table R for which the consistency problem is co-r.e.-complete, that is: (a) given a program P for a Turing machine, and initial data $d_1 d_2 \dots d_n$ for the program P, this can be mechanically converted to an initial configuration C such that*

There is no extension of C to the whole plane
$\Leftrightarrow$
The program P when run with input $d_1 d_2 \ldots d_n$ eventually halts;

and (b) there is a Turing machine M which, when run on input a partial configuration C (suitably encoded), will run forever just in case that C can be extended to the whole plane following rules in R.

Corollary 3. *There are rule tables R for which the consistency problem has no algorithm solving it.*

In many respects, this result is rather like the analogous result for tiling problems with initial tile [7, 4]. The question there is, given a set of finitely many types of square tile with coloured edges, can the plane be tessellated by these tiles in such a way that only edges of the same colour may touch (and, in some versions of the problem, a given tile or finite pattern of times is included somewhere)? The differences are in the fact that these tiles have an 'orientation' which may be used to simulate a Turing machine, whereas on the face of it, the squares in the infinite minesweeper problem do not.

However, both the infinite minesweeper problem and the tiling problem can be shown to be co-r.e. by the same method.

Proposition 4. *The general consistency problem is co-r.e., i.e., there is a Turing machine M which, when run on input a rule table R and a partial configuration C (suitably encoded), will run forever just in case that C can be extended to the whole plane followng rules in R.*

Proof. This is an application of König's lemma. The important thing to note is that given an $n \times n$ grid, each square being labelled by a single symbol from the finite alphabet Σ there are only finitely many ways to extend this to an $(n+2) \times (n+2)$ grid by adding a new strip of labelled squares all round. (To be specific, there are k^{4n+4} many ways, where k is the cardinality of Σ.)

Now form a finitely branching tree of all $k \times k$ grids which contain the initial configuration C in the centre, and do not directly contradict the rules R—where the adjacency relation on the tree is that of one configuration being in the middle of another. König's lemma tells us that either this tree is finite—i.e., there is some k such that *no* $k \times k$ square can contain C in the centre without directly violating R—or else the tree is infinite—i.e., the whole of the plane can be covered with labeled squares obeying the rules in R.

This noted, we design our Turing machine to search for k such that no $k \times k$ square can contain C in the centre without directly violating R. If the machine finds such a k it halts immediately, and if not it goes on for ever. □

We now prove the completeness, that Turing machine computations can be reduced to infinite minesweeper problems. I first need to explain a technical lemma used to get things started.

When we consider our infinite square grids, it will often be convenient to think of the squares as being coloured like a chess board, with two alternating colours that only touch at the corners of the squares. The lemma says that a periodic pattern on a square grid *with only the orthogonal adjacency relations considered* can be recovered from a rather natural rules table for the infinite minesweeper game which determines the pattern on alternate squares

(i.e., squares of the same colour) rotated at 45 degrees. Before I attempt to state it, we will consider an example first.

Example 5. *Consider the symbols $a, b, c, d, e, f, g, h, i$ arranged in a periodic pattern as follows.*

$$\begin{array}{cccccccc}
 & \vdots & \vdots & \vdots & \vdots & \vdots & & \\
\cdots & c & a & b & c & a & \cdots & \\
\cdots & f & d & e & f & d & \cdots & \\
\cdots & i & g & h & i & g & \cdots & \\
\cdots & c & a & b & c & a & \cdots & \\
\cdots & f & d & e & f & d & \cdots & \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & &
\end{array}$$

Now let Σ be a set containing $\{a, b, c, d, e, f, g, h, i\}$ and consider a rules table R for Σ as follows

	a	b	c	d	e	f	g	h	i	$\cdots$
a	0	1	1	1	0	0	1	0	0	$\cdots$
b	1	0	1	0	1	0	0	1	0	$\cdots$
c	1	1	0	0	0	1	0	0	1	$\cdots$
d	1	0	0	0	1	1	1	0	0	$\cdots$
e	0	1	0	1	0	1	0	1	0	$\cdots$
f	0	0	1	1	1	0	0	0	1	$\cdots$
g	1	0	0	1	0	0	0	1	1	$\cdots$
h	0	1	0	0	1	0	1	0	1	$\cdots$
i	0	0	1	0	0	1	1	1	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

which is obtained by putting a one for each orthogonal adjacency in the previous grid and a zero in other cases. The entries for other symbols in Σ in R can be anything we like. Then possible configurations of the whole plane include those of the form

$$\begin{array}{cccccccccc}
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \\
\cdots & ? & c & ? & g & ? & e & ? & \cdots & \\
\cdots & f & ? & a & ? & h & ? & f & \cdots & \\
\cdots & ? & d & ? & b & ? & i & ? & \cdots & \\
\cdots & g & ? & e & ? & c & ? & g & \cdots & \\
\cdots & ? & h & ? & f & ? & a & ? & \cdots & \\
\cdots & b & ? & i & ? & d & ? & b & \cdots & \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & &
\end{array}$$

and rotations and reflections of this, where the ?s are some symbols in Σ not in $\{a, b, c, d, e, f, g, h, i\}$.

It is not immediately obvious to me at present whether for the example just given *all* configurations of the plane are of the form shown, in other words, whether the original periodic grid can be recovered up to rotations and relections. But for 5×5 patterns and larger, this *is* the case, and this is the content of the next lemma.

Lemma 6. *Let Δ be a set of $n \times m$ district symbols arranged in a periodic pattern as above, where n and m are both at least 5; let $\Sigma \supseteq \Delta$ be a finite set of symbols, and suppose a rules table R for Σ is given such that the entry for $(\alpha, \beta) \in \Delta^2$ is 1 is α and β are orthogonally adjacent in the periodic pattern, and 0 otherwise. Then the only possible configurations of the whole plane obeying R have the symbols in Δ on alternate squares of one colour, symbols from $\Sigma \setminus \Delta$ on the squares of the other colour, and the symbols in Δ are in the same periodic pattern (rotated through $45°$) or possibly a rotation or reflection of this pattern.*

Proof. Suppose $a, b \in \Delta$ are orthogonally adjacent in the original pattern then they will be adjacent in the minesweeper configuration at the end. Suppose the original pattern is

$$\begin{array}{cccc} \cdots & x & c & \cdots \\ z & a & b & d \\ \cdots & y & e & \cdots \end{array} .$$

Then b is also adjacent to c, d, e. If a and b are *orthoganally adjacent* in the minesweeper configuration at the end then this forces the positions of the neighbours c, d, e of b to be as in

$$\begin{array}{ccc} ? & ? & c \\ a & b & d \\ ? & ? & e \end{array}$$

or various symmetric variations of this. (This is because $d \neq z$ since $n, m \geq 4$ and hence none of c, d, e can be neighbours to a.) But configurations like this are impossible as no two of c, d, e are allowed to be adjacent. This proves the symbols from Δ lie on all the squares of one colour, and the squares of the other colour must contain symbols from $\Sigma \setminus \Delta$ only.

It is now possible to see that the periodical pattern must be preserved, since if not we will have a configuration like

$$\begin{array}{ccc} ? & b & ? \\ a & ? & d \\ ? & * & ? \end{array}$$

with $*$ being from Δ and all the ?s being from $\Sigma \setminus \Delta$. But if the original pattern is 5×5 or greater, there is no such symbol from Δ adjacent to both a and d. □

In the applications of this lemma in this paper, I will take periodic patterns of size $n \times m$ where n, m are in the hundreds or thousands. But nevertheless it seems interesting to see if the number 5 in the lemma can be reduced to 4 or 3. (I suspect it can be reduced, at least to 4.) Note that in the argument in the second paragraph of the proof, in the case of a 4×4 pattern, the character z would be a candidate for such a symbol at $*$ adjacent to both a and d.

The periodic pattern in the lemma will be used as a sort of 'self-replicating' component of a machine, or vast logic circuit, simulating the action of a Turing machine. To be specific, each component will carry out the computation that determines the outcome of a single square of the Turing machine's tape at a single moment in time. If we can simulate each of these computations we will have succeeded in simulating the entire Turing machine computation.

In what follows, I will assume a certain amout of familiarity with Turing machines and the proof that the original minesweeper problem is NP-complete [5].

Our rectangular periodic 'machine component' will be designed as a complex logic gate taking various logical signals in and out of each side via devices like the wires used in the original minesweeper problem. We will take for Our Δ a finite set of symbols $\{a, b, c, d, e, f, \ldots\}$ which are arranged in a sufficiently large rectangular pattern, and for Σ the set $\Delta \cup \{0, 1, N\}$, where 0 represents 'off' or 'false', 1 represents 'on' or 'true', and N represents 'neutral'.

The rule table will include complete rows of ?s for $0, 1, N$:

	0	1	N	a	b	c	d	e	f	g	h	i	j	k	$\cdots$
0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	$\cdots$
1	?	?	?	?	?	?	?	?	?	?	?	?	?	?	$\cdots$
N	?	?	?	?	?	?	?	?	?	?	?	?	?	?	$\cdots$
$\vdots$															

The basic material making up our component is, like in the proof of the NP-completeness of ordinary minesweeper, wire. In the context of inifinite minesweeper, a *wire* is a minesweeper configuration of the form

$\cdots$	N	a_4	N	b_4	N	c_4	N	d_4	$\cdots$
$\cdots$	e_3	N	f_3	N	g_3	N	h_3	N	$\cdots$
$\cdots$	0	i_W	1	j_W	0	k_W	1	l_W	$\cdots$
$\cdots$	m_3	N	n_3	N	o_3	N	p_3	N	$\cdots$
$\cdots$	N	q_4	N	r_4	N	s_4	N	t_4	$\cdots$

Where the subscript $_4$ indicates that the rule table for this symbol says there should be four Ns adjacent to it, the subscript $_3$ indicates that there should be three Ns adjacent to it, and the subscript $_W$ indicates that there should be two Ns, one 1 and one 0 adjacent to it. Thus a fragment of the rule table for the wire above would be,

	0	1	N	a	b	c	d	e	f	g	h	i	j	k	l	m	n	$\cdots$
a	?	?	4	0	0	0	0	1	1	0	0	0	0	0	0	0	0	$\cdots$
b	?	?	4	0	0	0	0	0	1	1	0	0	0	0	0	0	0	$\cdots$
c	?	?	4	0	0	0	0	0	0	1	1	0	0	0	0	0	0	$\cdots$
d	?	?	4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	$\cdots$
e	?	?	3	1	0	0	0	0	0	0	0	1	0	0	0	0	0	$\cdots$
f	?	?	3	1	1	0	0	0	0	0	0	1	1	0	0	0	0	$\cdots$
g	?	?	3	0	1	1	0	0	0	0	0	0	1	1	0	0	0	$\cdots$
h	?	?	3	0	0	1	1	0	0	0	0	0	0	1	1	0	0	$\cdots$
i	1	1	2	0	0	0	0	1	1	0	0	0	0	0	0	1	1	$\cdots$
j	1	1	2	0	0	0	0	0	1	1	0	0	0	0	0	0	1	$\cdots$
k	1	1	2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	$\cdots$
l	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	$\cdots$
m	?	?	3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	$\cdots$

In the remaining diagrams, I will generally omit the symbol from Δ and just use the suffix 3, 4, W, or whatever, it being understood that each of these suffices should be labelled with a unique symbol from Δ and the appropriate row in the rule table constructed accordingly. What's more, the suffices 3 and 4 seem to be fairly ubiquitous and rarely contribute to the workings of the components, so will be usually ommitted: a blank square where a symbol from Δ is expected

will denote either 3 or 4, the context making it plain which is required. Similarly, a blank square where one of $0, 1, N$ is expected, will denote N, as these are very common and rarely contribute to the workings of the component.

As in the NP-completeness proof for the ordinary minesweeper game, we will now present methods for manipulating wires and building logic gates.

As one might expect, one can easily split, bend, start and terminate a wire. There are two ways to start or terminate a wire: either by specifying precisely the value (0 or 1) the wire should have at that point, or by allowing both possible values. The latter of these is achieved by the 3 suffix as will be shown in a moment. The former require new suffices T (true) and F (false) which have rule tables as follows

	0	1	N	$\cdots$
T	0	1	3	$\cdots$
F	1	0	3	$\cdots$

The following diagram illustrates splitting and bending wires and all three kinds of terminator.

									N	
0	W	1	W	0	W	1	W	0	F	N
	N	W	N						N	
		0								
	N	W	N				N			
	3	1	W	0	W	1	T	N		
	N	W	N				N			
		0								
	N	W	N				N			
	3	1	W	0	W	1	3	N		
	N	3	N				N			

The next diagram represents a NOT gate, or could be used to change the phase of a signal in a wire from $0101\ldots$ to $1010\ldots$. The line of the rule table for a new suffix $\neg$ is given first, and an example of its use afterwards.

	0	1	N	$\cdots$
$\neg$	2	2	0	$\cdots$

				4	N	3	N	4			
			3	N	W	1	W	N	3		
0	W	1	W	0	$\neg$	0	W	1	W	0	
			3	N	W	1	W	N	3		
				4	N	3	N	4			

The next gate I call an IF gate, and has the properties that if the input is 1 the output must be 1, but if the input is 0 the output can be 0 or 1. To make one of these, we need two more suffices, 2 and I as follows:

	0	1	N	$\cdots$
2	?	?	2	$\cdots$
I	2	1	1	$\cdots$

From this we can construct an IF gate, given here in its three possible configurations. First, with input true and output true:

$$\begin{array}{ccccccccccc} & & & & & N & 4 & N & & & \\ & & & & N & 3 & N & 3 & N & & \\ 1 & W & 0 & W & 1 & I & 0 & W & 1 \\ & & & & N & 2 & 0 & 2 & N & & \\ & & & & & N & 3 & N & & & \end{array}$$

Next, with input false and output true:

$$\begin{array}{ccccccccccc} & & & & & N & 4 & N & & & \\ & & & & N & 3 & N & 3 & N & & \\ 0 & W & 1 & W & 0 & I & 0 & W & 1 \\ & & & & N & 2 & 1 & 2 & N & & \\ & & & & & N & 3 & N & & & \end{array}$$

Finally, with input false and output false:

$$\begin{array}{ccccccccccc} & & & & & N & 4 & N & & & \\ & & & & N & 3 & N & 3 & N & & \\ 0 & W & 1 & W & 0 & I & 1 & W & 0 \\ & & & & N & 2 & 0 & 2 & N & & \\ & & & & & N & 3 & N & & & \end{array}$$

The idea of the IF gate is that it enables us to define the usuual AND, OR, NAND, NOR, and XOR gates. For example, if we draw up a table of all the possible logical values for A, B, C and IF(C) and then check which rows have exactly two values true out of these four we get,

A	B	C	IF(C)	Count = 2?
0	0	0	0	
0	1	0	0	
0	0	0	1	
0	1	0	1	yes
1	0	0	0	
1	1	0	0	yes
1	0	0	1	yes
1	1	0	1	
0	0	1	1	yes
0	1	1	1	
1	0	1	1	
1	1	1	1	

We see that the value for C in the four 'correct' lines is always $\neg(A \vee B)$, i.e., this can be used to define a NOR gate.

Using the suffix G with rule,

	0	1	N	$\cdots$
G	2	2	0	$\cdots$

we can give a configuration for a minesweeper NOR gate, writing a, b, c for the truth values of the wires A, B, C, a', b', c' for their negations, and x, x', y (where

x is the value of IF(C)) for the additional truth values required:

a	W	a'	W	a	W	a'													
				3		W													
			2		2	a	2		3		W	c'	W	c	W	c'	W	c	
		W	b'	W	b	G	c	W	c'	W	c	W		W					
		b	W		2	x	2		3		2		3	c'					
		W		4		W	N	3	N	W	c	W		W					
b	W	b'			3	x'	I	c	W	c'	$\neg$	c'	W	c					
					N	2	y	2	N	W	c	W							
						N	3	N											

Note particularly that from the 'suffices' $2, 3, 4, W$, etc., all the positions of the 'neutrals' N, including those not marked above are determined by the symbols from Δ alone, so the position of the wires are exactly as shown on the diagram above.

Here is the gate just given in the case when A and B are false and C is true:

0	W	1	W	0	W	1													
				3		W													
			3		2	0	2		3		W	0	W	1	W	0	W	1	
		W	1	W	0	G	1	W	0	W	1	W		W					
		0	W		2	1	2		3		2		3	0					
		W		4		W	N	3	N	W	1	W		W					
0	W	1			3	0	I	1	W	0	$\neg$	0	W	1					
					N	2	0	2	N	W	1	W							
						N	3	N											

As is well-known, from NOR gates and NOT gates, all the other standard logic gates can be constructed; *planar* circuits to do this were discovered by Goldschlager [3], and a way to do this is indicated my the original paper on the NP-completeness of *Minesweeper*.

For the final part of the proof that Turing machine computations can be reduced to infinite minesweeper games, consider a Turing machine, M. Without loss of generality, we can assume it has a single tape, a single read/write head, it has working alphabet $\{0, 1\}$, where 0 represents blank, and at most $2^k - 5$ states for some natural number k, which will be denoted by the integers $5, 6, 7, \ldots, 2^k - 1$. (The 'states' $0, 1, 2, 3, 4$ will be given a special meaning in a moment.) State 5 is taken to be the initial starting state of the Turing machine, and the 'transition function' of the machine is a *partial* function $T\colon (d, s) \mapsto (d', s_L, s_R)$ where $d, d' \in \{0, 1\}$ are symbols on the tape, $5 \leq s \leq 2^k - 1$ is a state, and either $s_L = 0$, $5 \leq s_R \leq 2^k - 1$ (indicating a move right entering state s_R, or $s_R = 0$, $5 \leq s_L \leq 2^k - 1$ (indicating a move left entering state s_R).

We design a rectangular 'logic gate' with inputs and outputs as shown in Figure 1. The 'data in' and 'data out' arrows indicate a single wire carrying a value 0 or 1, representing the data on the Turing machine's tape at this point. The various 'state in' and 'state out' arrows indicate k wires, carrying data for a single number from 0 to $2^k - 1$ in binary, and represent the Turing machine's state and its head's position, as will be described in a moment. The idea is that this component is defined by the symbols in Δ which repeat periodically (at 45° to the normal grid, but this isn't a problem), so that similar components tile

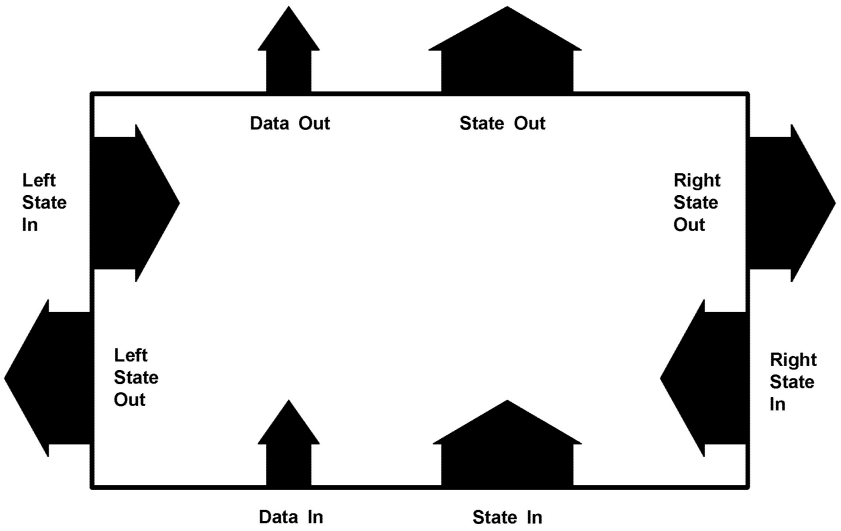


Figure 1: A Minesweeper component simulating a Turing machine

against one another with the 'state out' of one component being the 'state in' of the next, and similarly for the other arrows.

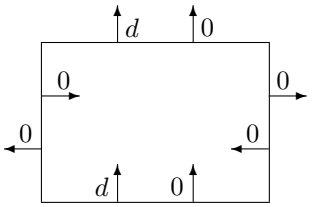
The remaining details concern how to program a single component so that in total they behave like our Turing machine acting on a tape initially containing the data $d_1 d_2 \ldots d_n$.

Actually, the component doesn't quite work like a straightforward logic gate with inputs and outputs as implied by the diagram; it has a 'nondeterministic' feature, built using the IF gate, and also in certain cases it verifies the inputs are the expected values—it cannot be used if this turns out not to be the case. (For a simple example of a component with this last feature consider a terminator for a wire ending in 'true': it can only be used if its input has the correct value. A more complex example of this is seen in the method used to compute an arbitrary propositional formula and verify it true, in the proof that ordinary minesweeper is NP-complete.)

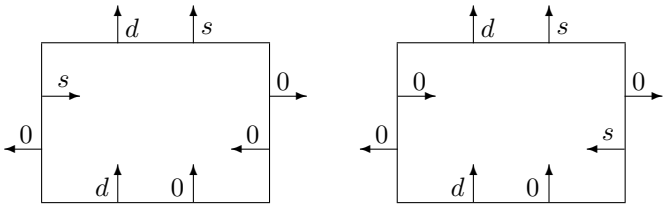
The component's job then is to verify that its inputs and outputs are of one of the following eleven forms. (My explanations as we go along will explain what these forms do and the meaning of the mysterious states $0, 1, 2, 3, 4$.)

Firstly, state 0 is used to indicate that 'the Turing machine's head is not here at present'. If this is the case, the data on the tape must not change. Our

first possible form is therefore

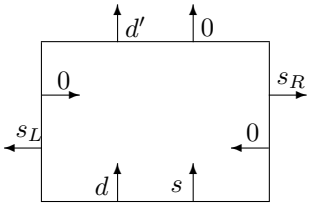


Next, if a state $s \geq 5$ is provided from the left or right, that means the Turing machine's head has moved to this square. The data doesn't change yet, but we must indicate the presence of the read/write head. This gives two more possible forms:



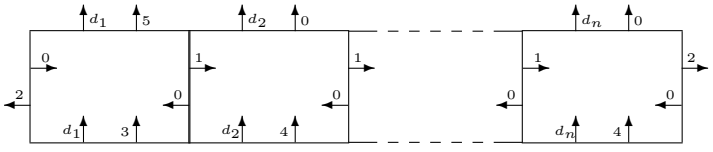
for $d \in \{0, 1\}$ and $s \geq 2$.

The next possible form encodes the transition function of the machine. For each possible $T\colon (d, s) \mapsto (d', s_L, s_R)$, *provided neither* s_L *nor* s_R *is the 'halt' state* we have



Observe that if the Turing machine head is at this square it won't be anywhere else, so we can reasonably check that 'state left in' and 'state right n' are zero.

The next group of possible forms concern the starting position of the Turing machine at time 0. We will be looking for configurations covering the whole plane that contain the following starting position built from n components tiled next to one another:



The idea is that the new 'states' $1, 2, 3, 4$ mean the following:

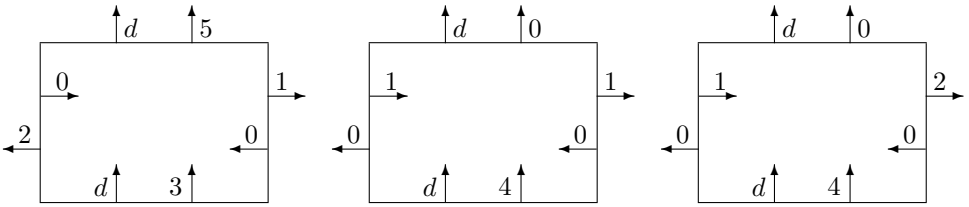
1: the square to the right contains initial data for the Turing machine;

2: all squares in this direction are to be set to zero;

3: the Turing machine head is here, but the machine hasn't started yet;

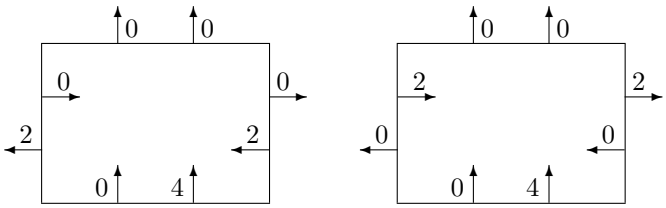
4: the Turing machine head is not here, and the machine hasn't started yet.

To allow the starting position just indicated our component must admit the following three forms

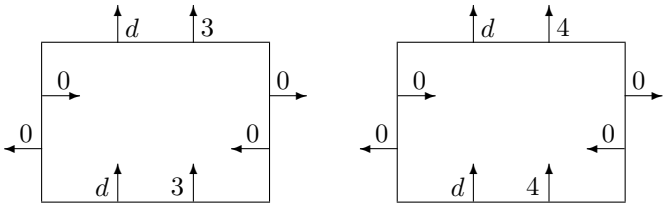


for each of the two possible values of d, 0 or 1.

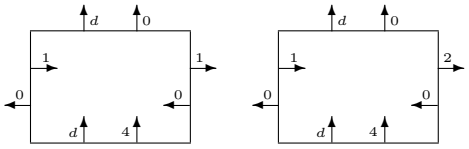
To clear to zero the remaining parts of the tape to the left and right we need to allow



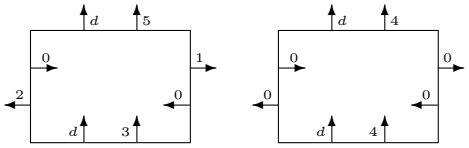
The remaining two forms we need concern what happens to the Turing machine 'before it has started'. In these cases we just make sure the head doesn't move and the data on the tape stays the same. We allow, for each of the two possible values of d the forms.



These eleven are the only forms allowed by the Minesweeper 'logic component'. Notice the two places the IF gate idea was needed in building the component: both



were needed to accommodate the initial data, and



were needed to allow the Turing machine to start (and so that the whole plane could be covered).

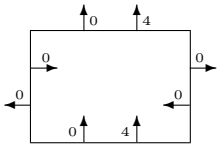
Notice also that we purposely did not allow any forms that involved the machine entering any 'halt' state, so the *only* way the whole plane can be filled with these configurations, including the start configuration indicated above, is if the Turing machine M when run on input $d_1 d_2 d_3 \dots d_n$ goes on for ever. (There are some things to check here, to make sure that there are no ways of covering the whole plane by some method that wasn't intended, but I'll leave the fun verifying this to the reader!)

Conversely, the plane can be filled with such configurations and including the start configuration if it *does* go on for ever.

This proves the theorem claimed earlier, and also its corollary since it is known (by using a universal machine) that there are machines M with working alphabet $\{0, 1\}$ such that the halting problem for this machine is algorithmically undecidable.

Note that the last part of the proof just given is essentially just the reduction of the halting problem to a particular form of the tiling problem due to Wang [7]. Wang showed that it is algorithmically impossible to determine if a given set of tiles can tile the plane *given that one particular tile must be used*, and that there is a single set of tiles for which it is algorithmically impossible to determine if a given set of tiles can tile the plane *given that a particular finite configuration is used somewhere*.

A much sharper form of the tiling problem was proved undecidable by Berger in 1966 when he showed that the first tiling problem just mentioned in the previous paragraph for a general set of tiles is still algorithmically unsolvable even if we don't make the restriction that a particular tile must be used. (Grünbaum and Shepherd [4] give a good presentation of these issues.) Note that in the presentation just given, the configuration



is valid and, on its own, does tile the plane.

The analogous question for infinite minesweeper is whether

The general consistency problem for rules: on input a set of rules R, determine if there is a complete labelling of the whole plane that obeys the rules in R.

is algorithmically unsolvable. This question can be answered directly by methods just given, by using a reduction of the form of the tiling problem in Berger's theorem to infinite minesweeper. Specifically,

Theorem 7. *The general consistency problem for rules is co-r.e.-complete.*

This theorem is proved by showing that, given a set of tiles, they can be mechanically converted to a rules-table such that

> The plane can be labelled by symbols in the table following the rules
> $\Leftrightarrow$
> The plane can be tiled by the set of tiles.

Concluding remarks. One small point about this result that I find strange is that (A) the consistency problem for finite minesweeper is NP-complete, even for a very general set of rules, not just the normal rules whereas, as just proved, (B) the consistency problem for infinite minesweeper is *co-r.e.*-complete. This seems to disagree with the common wisdom that NP and *r.e.* are analogous, as are co-NP and *co-r.e.*. I don't have any further comment to make on this though. (I certainly wouldn't dare suggest on this evidence alone that NP=co-NP!)

An obvious problem left over from this work is to find a reasonably simple table of rules that is algorithmically unsoluble, in the sense of Theorem 2. It is perfectly possible and in principle straightforward to extract from the description I have just given a algorithmically insolvable set of rules for the infinite minesweeper problem. But, at a quick estimate, there would be several thousand symbols, and a rule table of over a million entries. In contrast, the beautiful feature of *Life* is that its rules are so simple. It would be very nice (and a testing challenge for designers of mathematical games like this) to have a simple, memorable set of rules for which the consistency problem is unsolvable.

References

[1] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning ways for your mathematical plays. Vol. 2, Games in particular.* Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1982.

[2] Martin Gardner. *Wheels, life and other mathematical amusements.* W. H. Freeman and Co., San Francisco, Calif., 1983.

[3] L. M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–29, 1977.

[4] Branko Grünbaum and G. C. Shepherd. *Tilings and Patterns.* Freeman, New York, 1987.

[5] Richard Kaye. Minesweeper is NP-complete. *Mathematical Intelligencer*, 22(2):9–15, 2000.

[6] Hao Wang. *Popular lectures on Mathematical Logic.* Van Nostrand, New York, 1981.

[7] Hao Wang. *Popular lectures on Mathematical Logic.* Dover, New York, 1993. Enlarged republication of an earlier work of 1981.