

Commuters and Computers: the Intelligent Subway

Ian Stewart

24/09/2008

**Mathematics Institute
University of Warwick
Coventry CV4 7AL
UK**

"Well, if the universe *is* algorithmic —"

"— As many scientists believe —"

"— then strong AI —"

"— don't be pedantic, Dee, what you mean is computers that think — "

" — *must* be possible in principle."

The Tweedle twins were strap-hanging on the New York subway, but wherever they were and whatever they were doing, their conversations were always similar. They perpetually interrupted each other.

Delia Tweedle was universally known as Dee, so her brother had inevitably become Dum. Even though his real name was Seymour.

"Why?" I said.

"Because if our universe is algorithmic—"

"— you could set up a computer to simulate it —"

"— which would therefore simulate everything in it, including us having this conversation," Dee concluded triumphantly.

"Which you think is an outstanding example of intelligence?"

"Well, choose your own example of intelligence," said Dee, looking offended. "The same argument holds."

"What do you mean by the universe being 'algorithmic'?" I asked.

"Obeying underlying mathematical laws," said Dum.

I thought about that for a few moments. "You realise that if you're right, then a sufficiently complex subway system could become intelligent?" I said. "It would think rather *s-l-o-w-l-y*... but it would still be able to think. If you're right," I clarified nervously, as everybody else in the car stared at me as if I was crazy. Considering what a weird bunch they were, I felt it was *my* turn to look offended. The train stopped at Rockaway Boulevard and most of the passengers got out. We sat down and admired the graffiti.

"That's dumb," said Dee. "A subway can't think."

"Maybe not — but a subway can certainly compute."

"A *subway*? Compute? Science fiction," said Dum.

"Funny you should mention that," I replied. "Offhand, I can only think of two science-fictional subway stories. One is A.J.Deutsch's *A Subway Named Möbius*, in which a subway system becomes so topologically complicated that the trains on it cease to have well defined locations. The other is Colin Kapp's *The Subways of Tazoo*, in his

'unorthodox engineers' series. But what I have in mind is more like David Brin's *Earth*, where the global computer network become so interconnected that it acquires enough intelligence to save humanity from a rogue black hole."

"You're straying—" said Dum.

"— from the point," concluded Dee.

"No, I'm not, really," I said. "Your discussion reminded me of a fascinating article I've just read in the latest issue of *Eureka*, the journal of Cambridge University's mathematical society, the Archimedean."

"Cambridge Mass?"

"No, Cambridge UK. It was written by Adam Chalcraft and Michael Greene, and it's all about the computational abilities of train sets."

"Some special sort of set theory?" hazarded Dum. "Related to neural networks? 'Train' as in 'teach'?"

"No, not that sort of train. Train as in 'Oriental Express'."

"You mean *toy* train set? Rails and points and little men in out-of-date railroad uniforms?"

"That's right, Dee. And whatever a train set can do, a subway surely can."

"Train sets," said Dee, shaking her head. "Not quite your Hypercube parallel processor."

"In speed, no," I said. "But in theoretical computing ability, equally powerful. Remember, Alan Turing proved that any programmable digital computer can simulate any other, given enough memory. A computer, after all, is just a huge switching circuit with adaptable switches. And trains can switch tracks using points. What Chalcraft and Greene asked was, if you've got a big enough stock of straight and curved track, bridges, and various kinds of points — most important, those — but you've got only one engine and *no* rolling stock, then what computations can you do if you set up the right track layout?"

"I don't see how a train can compute at all," said Dee. "It's just a thing on wheels that moves along the rails."

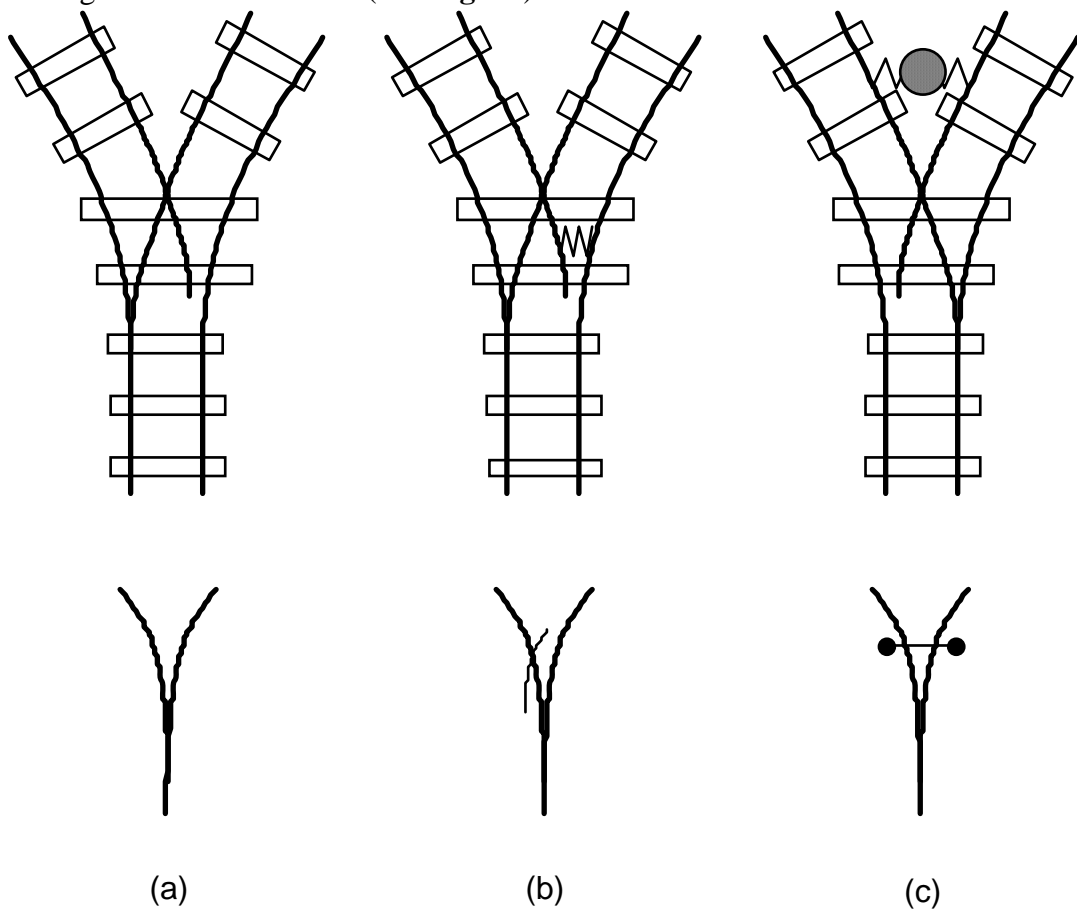
"Electrons are just things that move along wires, but computers compute by using them," I pointed out. "The computational aspect is a matter of interpretation in both cases; the underlying mechanisms are much the same. The idea is to encode an input as a lot of 0's and 1's corresponding to the settings of various points. Then you run the train through the layout and of course those settings change. Which in turn alters the path of

the train. Eventually the train is sidetracked into a line leading to a terminal — the program 'stops' — and you read off the output from the settings of the same collection of points."

"OK, I see that it might work, But does it?"

We pulled into Euclid Avenue and a teenager with dreadlocks and a bright green electric guitar pushed his way on board. He pulled a fat paperback out of his pocket and started to read. The title was *Principia Mathematica* by Russell and Whitehead. People are unpredictable.

I rescued my mind from its fruitless wanderings. "Let me start with the simplest switching unit, known as *lazy points*. They're a Y-shaped piece of track. A train entering the Y from below runs up the upright and out of whichever arm of the Y the points are set for, leaving them set as they were to begin with. However, a train that enters from one of the arms will — if necessary — reset the points so that they connect that arm to the upright, and exit via the upright. But no trains ever come in down one arm and go out via the other." (See **Fig.1a**.)



Three types of points (above) and their 'circuit symbols' (below). (a) Lazy points, (b) Sprung points, (c) Flip-flop.

"Lazy points have two states, depending on which arm is connected to the upright: let me call them *left* (L) and *right* (R).

"Layouts whose only active components are lazy points have very limited computational ability. Assume that you use a finite number of bits of track, fix some layout, and let the train run through it. The only things that change are the points, and these can be in only two states, L or R. If there are n lazy points then the layout has at most 2^n states. So eventually the motion of the train must fall into a repeating cycle, when it starts to run through a state of the layout that it has run through in the past. Discard any bits of track that aren't traversed during the cycle. Amazingly, there are only two topologically distinct types of layout that remain. One is just a closed loop with no points, and thus computes absolutely nothing at all. The other is a figure-8 arrangement with two points, in which the arms of each point are joined in a loop and the uprights are run together."

"I think 'points' is singular," objected Dum. "You should say 'pointses'." I gave him an unwavering glare. "OK, you've made your point," he said after a few seconds.

"Yeah. Now, suppose that in the second case the lazy points both start out in state R, with the train in between them. If you trace its path and the effect on the points you'll find that it cycles the settings in the sequence **RLRLLRLR**, repeated forever, where the bold symbols indicate one set of points and the plain letters indicate the other one. So it's like a computer that can only compute the terms of the sequence 10100101, repeated forever."

"You're letting each R represent 1 and each L represent 0?"

"That's right, Dee. For simplicity, I hasten to add. I'll clarify how to interpret the train's motion as a computation later, when we get to a more representative layout."

"Fair enough," said Dum.

"At any rate, now we know that we need some more complicated switching gear if we're going to build a computer. The next type of point is a *sprung* one (**Fig.1b**). It's like a lazy point except that any train entering along the upright of the Y always leaves by the same arm — say the left arm."

"Does it matter which arm you use?" asked Dum.

"Course not," yelled Dee. "By using bridges you can connect the rest of the layout to either arm so the right-armed sprung point doesn't do anything new."

"Right," I said. "But it does simplify layouts to some extent, so I'll allow either arm to be sprung. The third kind of point is a *flip-flop* (**Fig.1c**)."

"Good railroad jargon, that," said Dum. I ignored him.

"With a flip-flop, the train always enters along the upright of the Y, and exits through the left and right arms alternately," I said. We lurched judderingly into Liberty Avenue. "The big question is: given these components, can we build a computer?"

"In what sense?" asked Dee.

"A Turing machine will do," I said. "Alan Turing proved that his simple model of a computational system can do anything that a programmable digital computer can. Think of a Turing machine as a box which can travel along a very long tape of square cells, each containing either the symbol 0 or 1."

"How long?"

"As long as you need for a given computation, Dum. You can either use an infinite tape, or if you don't like infinities you just have to be prepared to add some more squares to the tape if you need them."

"Oh, right."

"The box can be in any of a finite set of internal *states*, depending on what hardware is inside it. For each combination of its own state and the digit on the tape immediately beneath it, it must obey a small list of instructions, like this:

- Leave the current tape digit alone / change it
- Then move one space left / right
- Then go into some specified internal state ready for the next step.

Alternatively, the instruction can be just 'stop' and the computation then finishes."

"Give me an example".

"OK. Here's one typical list of rules, with three states, 1, 2, and 3..

- *State 1, Digit 0*: Change digit, move left, go to state 2.
- *State 1, Digit 1*: STOP.
- *State 2, Digit 0*: Leave digit, move right, go to state 3.
- *State 2, Digit 1*: Change digit, move right, go to state 2.
- *State 3, Digit 0*: Change digit, move right, go to state 1.
- *State 3, Digit 1*: Leave digit, move left, go to state 2.

"What does that compute?"

"I haven't the foggiest idea. Try it and see. Sensible programs generally need a lot more states than three, anyway. Now, if there are T internal states then, in addition to 'stop', there are 4T possible instructions — two choices for the first, two for the second, and T for the third. You get one Turing machine for each assignment of rules to states.

"The digits on the tape provide the computer's input; the list of which instructions to perform for which internal state of the box form the program, and the list of digits on the tape when the computation stops is the output. Amazingly, these simple devices can carry out any algorithm whatsoever. So all we need is to find a train layout that simulates any chosen Turing machine."

"Tricky."

"Yes. It helps to break the problem down into a series of stages." We reached Rockaway Avenue and the teenager got out. "Now, the idea is to find a train layout that can play the role of the box. Then you just plug an enormous number of these boxes into each other, side by side, to represent the whole tape (**Fig.2a**). Each box will have T tracks coming in from the left and T going out at the right — one for each internal state."

"So instead of the box moving along the tape —" began Dee,

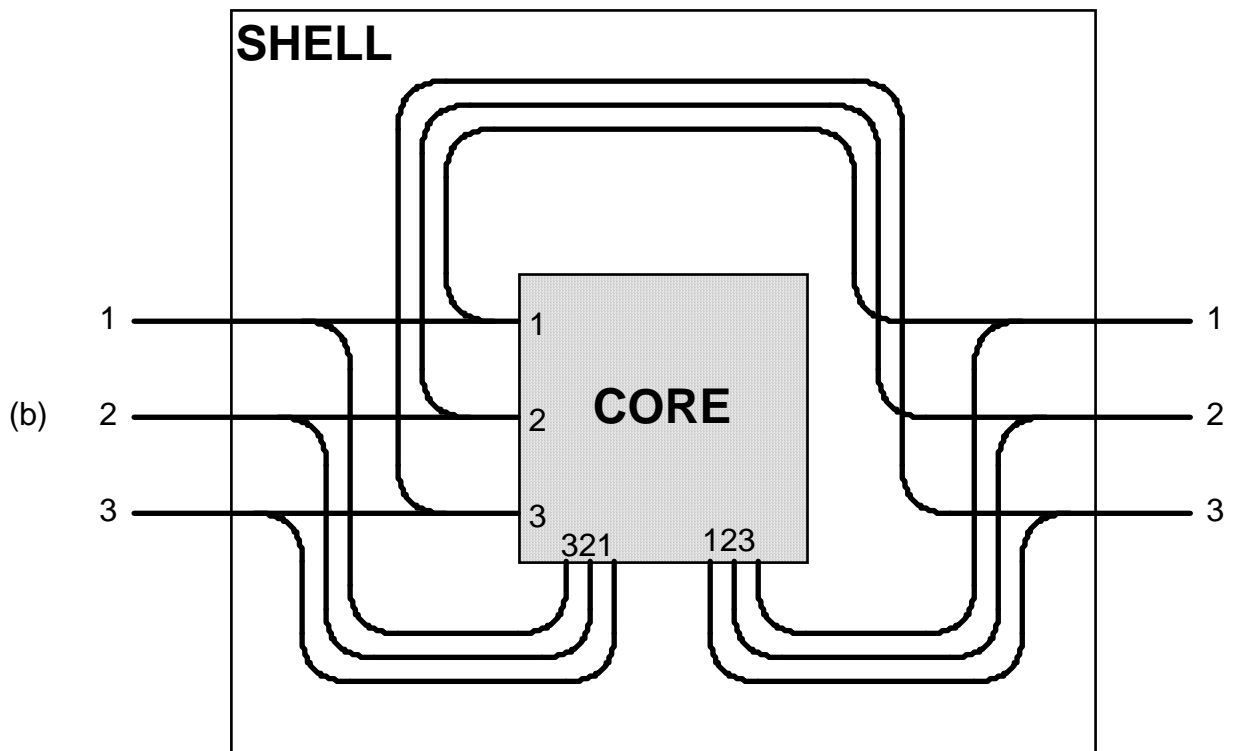
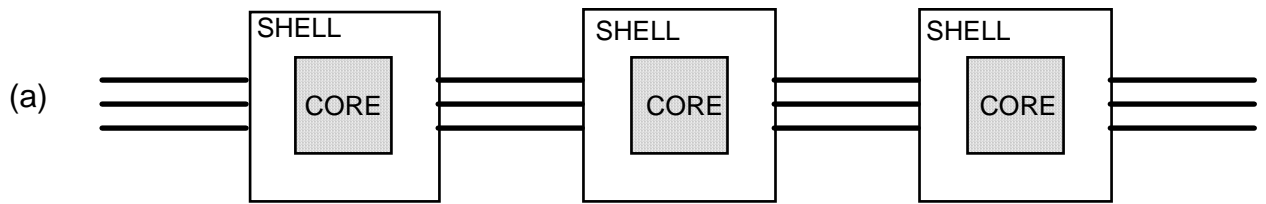
"— the train moves along the row of boxes," Dum finished, nodding enthusiastically.

"You can tell which 'square' of the 'tape' is being worked on —"

"— by which box the train is in. Neat."

"Yeah," said Dee. "But what do you put in the box? Schrödinger's cat?"

"I'll explain how the box is designed in stages. The train tracks are used as both input and output lines, so the box doesn't 'remember' which direction the trains came in from. So it can be set up as an outer shell that feeds trains the same way into an inner core from both input lines, and conducts them out again according to the Turing program." (See **Fig.2b**). "Then we can ignore the outer shell and just concentrate on the design of the core."



(a) Replacing the tape of a Turing machine by a series of identical track layouts. (b) Each square of the tape consist of a shell, which ensures that trains entering from either direction are treated alike, and a core, which simulates the rules for the Turing machine.

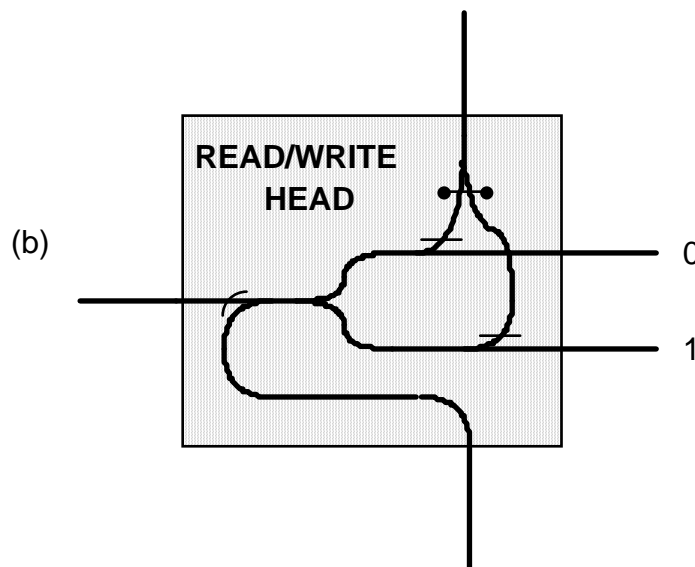
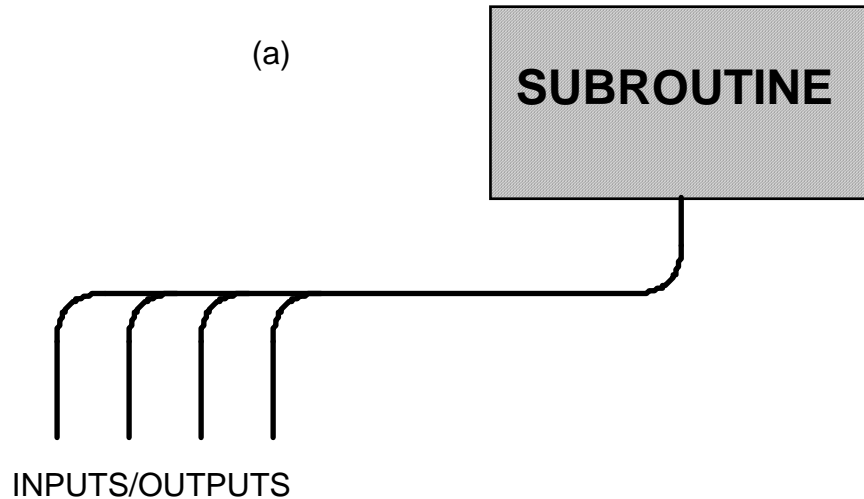
"You're going to need—" said Dum.

"— subroutines," said Dee. They'd been thinking ahead, as usual. A subroutine is a part of a program that can be used repeatedly by 'calling' it from any other part. You can build complex programs by stringing together subroutines. "Right," I said. "You can set up a subroutine by hooking up a self-contained sublayout to a whole series of lazy points. Then the train comes in, setting the points as it does so, wanders round the sublayout until it's carried out whatever subroutine that sublayout computes. Finally it exits by the same track that it came in on because of the way it set the points on entry. Using one lazy point for each input line, the trains can all be made to enter from the left,

carry out the subroutine, and exit to the right along the same track they entered from." See **Fig.3a**.

"Oh, right."

"Now, you need one more piece of gadgetry, a *read/write head* (**Fig.3b**).

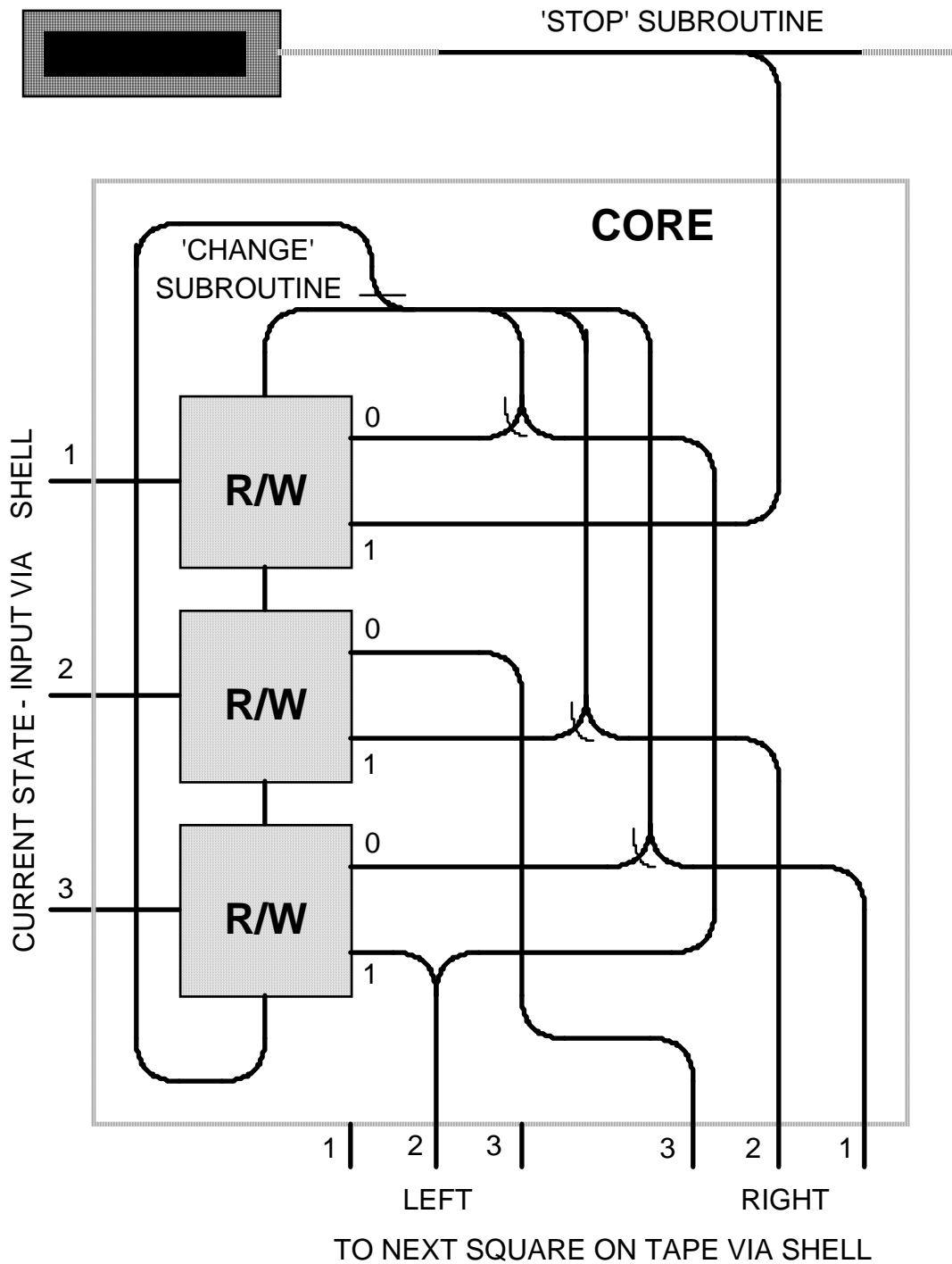


(a) Layout for calling a subroutine. Trains enter through lazy points, and exit along the same track. (b) Design of a read/write head: note the presence of a flip-flop.

If a train comes in from the left it exist along line 0 or line 1 depending on the digit at the 'current' point of the tape; and if a train comes in from above it swaps the 0 and the 1. To achieve this, the lazy point P is set to redirect the train along output lines 0 or 1 according to the digit on the 'tape' at that square. The flip-flop is set so that the first train entering from the top switches P to the other position.

"Having got all these bits and pieces, you build the inner core of the box as in

Fig.4.



Design of the core circuit. The train enters from the left, obeys the appropriate Turing machine rule, and exits at the bottom.

You may need some bridges to avoid the tracks crossing, but we can ignore that. The core consists of a parallel set of read/write heads, one for each internal state of the core.

The output lines 0 and 1 either lead to one of the outputs of the core, or to a lazy point that diverts the train into a subroutine that changes the state of that 'square' on the tape, or to a 'STOP' subroutine that guides the train into a single terminal.

"Let me see," said Dee. For your example one of the rules is '*State 1, Digit 0: Change digit, move left, go to state 2.*' How does that work?"

"The square being in state 2 means that the train enters from the side along line 2. This state is set by the output of the *previous* square, which directs the train on to line 2 when it exits. The digit 'written' on that square is 0: that just means that we've set all the lazy points in the read/write heads to 0. So the train comes into the second read/write head, leaves along line 0, and runs into a set of lazy points. These direct it into the 'change' subroutine. It runs vertically downwards, through all of the read/write heads, flipping their states from 0 to 1. So the digit 'written' on that square now reads 1, not 0. The train goes back up the vertical track to the left of the heads, exits from the subroutine back onto its original track, and then comes out of the core on the output line 2 *left*, which effectively moves the train into the box to the left, in state 2, as required."

"Cute. Let me see, suppose we look at the rule '*State 2, Digit 0: Leave digit, move right, go to state 3.*' The train comes in along line 2 and exits the read/write head along line 0, which leads directly to exit 3 *right*. And it never goes anywhere near the subroutine loop, so the state of the square remains unchanged."

"Right," said Dum. "And it's equally obvious that the rule '*State 1, Digit 1: STOP*' works properly. Enter along line 1, exit the read/write head along line 1, and you get fed straight into the line that ends at the terminal."

"Exactly. By setting up the lines according to the list of rules for the Turing machine, you can make the layout simulate that Turing machine exactly."

Dum and Dee admired Chalcraft and Green's clever design. Suddenly Dee nearly fell off her strap. I thought for a second that the subway had stopped, but it turned out she'd had an idea. "Do you really need bridges?"

"Eh?"

"Can't you find a substitute using only sprung points?" It turned out she was right — but I'll leave you the pleasure of working out how that particular trick can be done. "For that matter, can you build a flip-flop from lazy points and sprung points?" she asked, flushed with her previous success.

"It would be more elegant," said Dum.

"You wouldn't need those fancy flip-flop points," added Dee. They both nodded.

"That's a good question, " I said. "Chalcraft and Green find that it has a very curious answer. You can't build a flip-flop with a finite layout of lazy points and sprung points. But you can build it with an *infinite* layout." Again, I leave readers the pleasure of finding it. "But, it's only *just* infinite, in the sense that any finite computation only requires a finite portion of it."

"Just like the 'very long tape' you talked about earlier," said Dum.

"Do you realise," said Dee, that this shows that the future behaviour of a train-set can be undecidable?

"I see," said Dum. "Turing proved that the halting problem for Turing machines is formally undecidable. You can set up a Turing machine for which there is no way to decide, in advance, whether or not the computation stops."

"Which means that for the corresponding train layout, you can't predict in advance whether the train is ever going to reach the terminal."

"That's quite startling," I said. "I've never been terribly bothered about theoreticla questions of formal undecidability in computer science. But it's a bit worrying that you could set up a mechanical system with train tracks, whose workings are totally transparent, and not be able to answer such a simple question as whether the train will ever reach a chosen station."

"Speaking of which —" said Dee.

"— it's been an awfully long time since the last stop," said Dum.

"Shouldn't we have reached —"

"— Kingston-Throop by now?"

I wiped away the moisture that fogged the window. "Hmmm," I said. "All I can see is a big square with the digit '1' painted on it. And I swear there's a sign just along the tunnel that reads 'flip-flop 7743A/91.' "

"That's ominous," said Dee.

"Dee gets claustrophobic when subway trains stop," said Dum helpfully.

"They *have* been adding some new connecting lines to the subway network," I said. "Maybe its connectivity has passed the Turing Threshold and it has achieved intelligence..."

"O Mighty Subway," Dee declaimed, her voice rising rapidly in pitch, "we humble humans solicit your omniscient aid in *getting us the hell out of* —" At that moment a connecting door opened and a uniformed guard poked his head into the car.

"Small problem up the line, folks," he said. "Nothing to worry about, but we're slowing down for a few minutes while they clear up the mess." Dee sighed with relief. "Hey, is the little lady all right?"

"Yeah," said Dum. "She just thought she'd got stuck in an artificially intelligent Turing machine."

"This ain't no tourin' machine," said the guard. "This is just a personnel commuter, buddy."

At least, I *think* that's what he said.

FURTHER READING

Adam Chalcraft and Michael Greene, Train Sets, *Eureka* **53** (1994) 5-12. [To subscribe to this 'approximately annual' journal send £10 to the Business Manager, *Eureka*, Arts School, Bene't Street, Cambridge CB3 3PY, England. Internet address: archim@phoenix.cambridge.ac.uk.]

A.J.Deutsch, A subway named Möbius, in *Best SF 4* (ed. EdmundCrispin), Faber and Faber, London 1965.