Laver Tables and Busy Beavers

Peter Taylor

Work in progress 2016-06

Yedidia and Aaronson have provided by construction an upper bound on the largest two-symbol Turing machine for which ZFC can determine whether or not it halts when run on an initially empty tape[4]. Specifically, they construct a 7918-state machine which halts only if ZFC is inconsistent: to prove that it halts would be to prove ZFC inconsistent, but to prove in ZFC that it does not would also prove that ZFC is inconsistent, via Gödel's second incompleteness theorem. Unpublished work by Stefan O'Rear brought this limit down to around 1900 states[3].

In discussion around these results, Joseph Van Name proposed an open question about Laver tables as a candidate for reducing the limit still further. We present a Turing machine which implements this proposal: it can be proven to halt under the very strong assumption of a rank-into-rank cardinal, but no proof is known in ZFC. At 64 states, it is two orders of magnitude smaller than the Yedidia-Aaronson machine.

1 Laver tables

The Laver table A_n is the unique magma $(\{1, \ldots, 2^n\}, \star_n)$ satisfying $x \star_n 1 = x + 1 \bmod 2^n$ and $x \star_n (y \star_n z) = (x \star_n y) \star_n (x \star_n z)$. The sequence of $1 \star_n y$ is periodic with period a power of two, and Laver showed that assuming a rank-into-rank cardinal the period grows without bound[2]. However, even to show that there is a Laver table with period greater than 16 has thus far required a rank-into-rank cardinal hypothesis.

Direct application of the two properties of $\star_n$ allows computation of $x \star_n y$: where $y > 1$ we have $x \star_n y = (x \star_n (y - 1)) \star_n (x + 1)$, and the recursion eventually terminates (see appendix). However, for implementation in a Turing machine we need to simplify the tracking of the recursive state. By reversing the parameters we get a tail-recursive function

$$g_n(y, x) = \begin{cases} x + 1 & \text{if } y = 1 \\ g_n(x + 1, g_n(y - 1, x)) & \text{otherwise} \end{cases}$$

This gives rise to the following implementation in a stack-based pseudo-language:

```
while stack depth > 1:
        pop x
        pop y
        repeat y times:
                push (x+1) mod 2^n
```

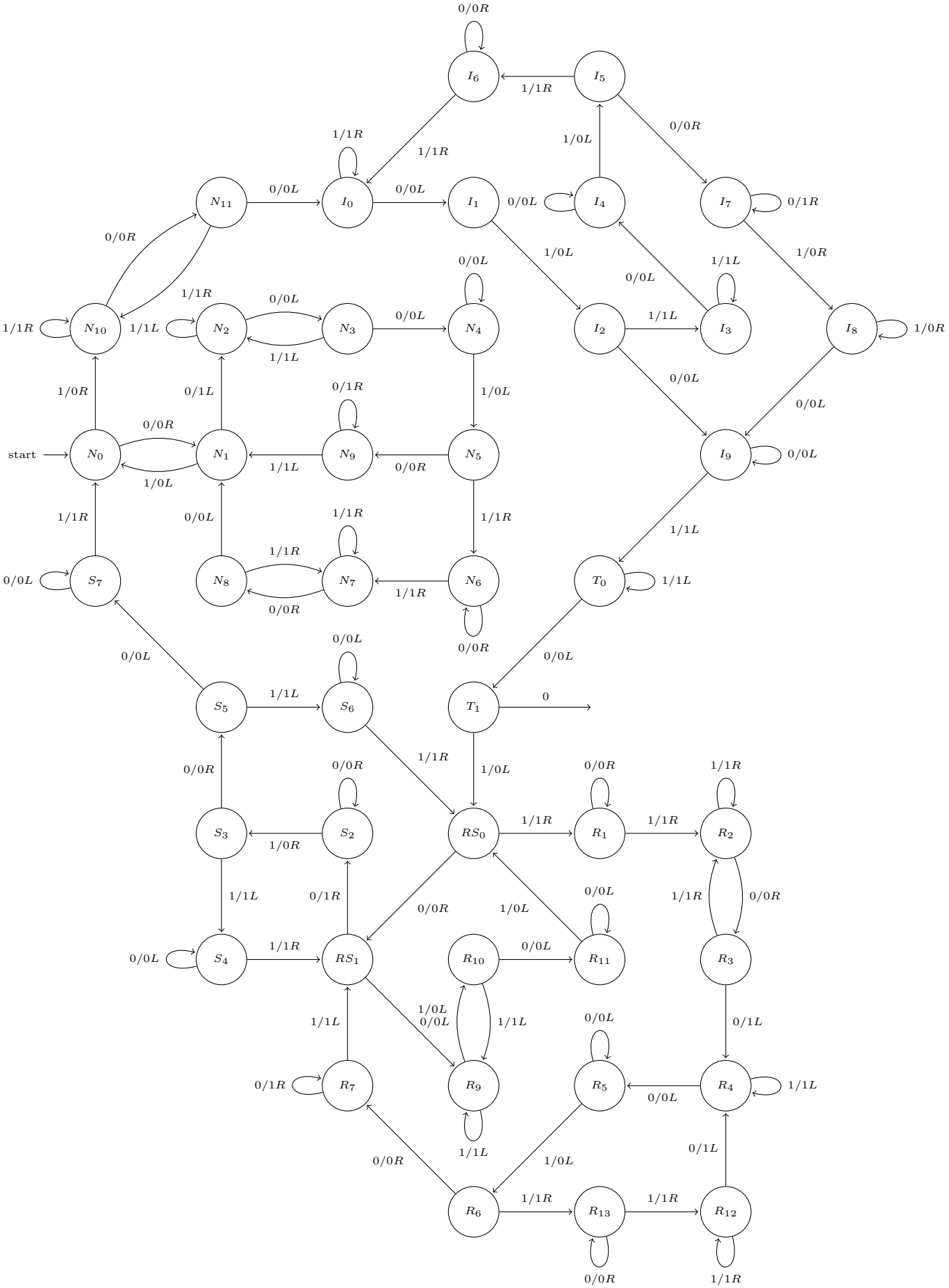
Since it will be necessary to use 2^n we will need an efficient way of locating it to copy it to the top of the stack in the implementation of $x \to x + 1 \pmod{2^n}$. The tape layout chosen is therefore to store 2^n in unary, followed by two 0 symbols, followed by the stack in unary, with each element separated by a single 0 symbol. The overall structure of the program is:

```
N = 8
repeat forever:
        N = 2N
        if (1 * 16) is not equal to N:
                halt
```

Note that N is initialised to 8 so that after the first doubling it will be 16 and we don't have to do a true modulo: the only value greater than N which will be encountered and must be reduced will be $N + 1$. But since we have that increment modulo N in the inner loop, and since comparison of a unary number to 1 is certainly going to be cheaper than comparison to N, once we fill in the details we can optimise to:

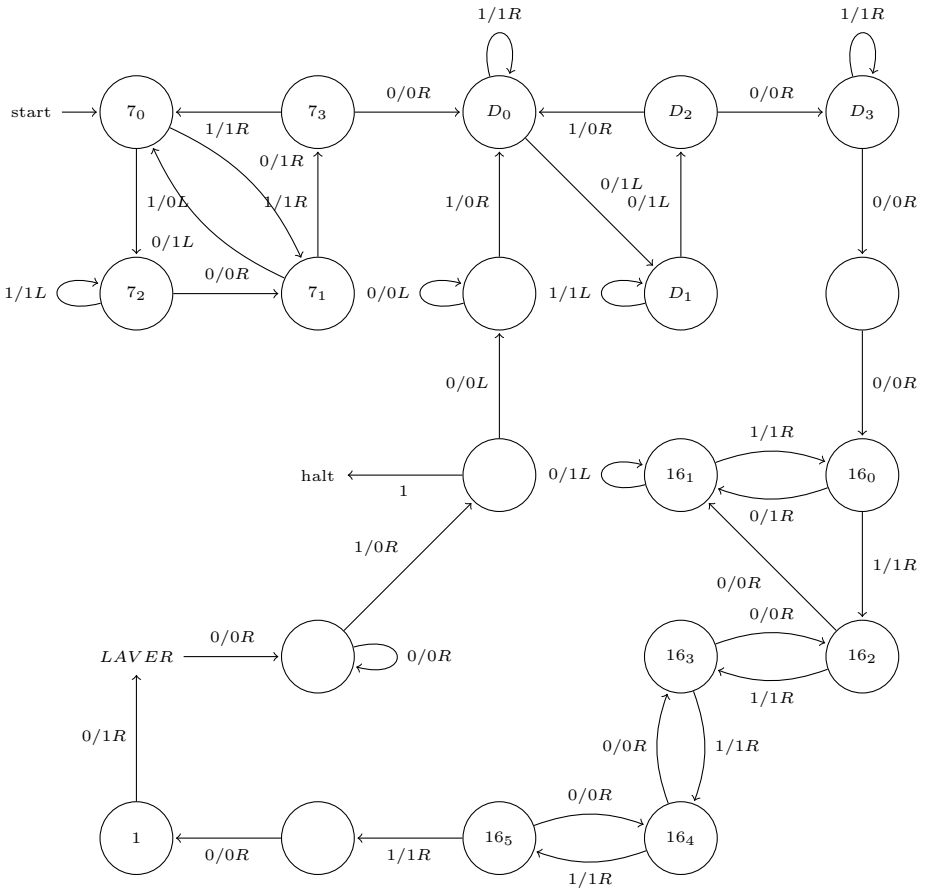
```
N = 8
repeat forever:
        N = 2N
        push 16
        push 1
        repeat forever:
                pop x
                push (x+1) mod N
                if stack depth = 1:
                        break from inner loop

                pop x
                pop y
                repeat y times:
                        push x

        pop x
        if x is not equal to 1:
                HALT
```

The first figure shows the 44 states which implement the inner loop. The states labelled N_i copy N from the bottom of the stack to the top; I_i map a top of stack holding x N to one holding $(x + 1) \bmod N$; T_i test the stack depth and break if we have a result; R_i replicate the top of the stack y times, deleting y, where y is the second value on the stack; and S_i slide those copies of $x + 1$ left to fill the gap left by y. The two states labelled RS_i are each aliased from states in two separate loops which had guaranteed input.


start
N_0
N_1
N_2
N_3
N_4
N_5
N_6
N_7
N_8
N_9
N_{10}
N_{11}
I_0
I_1
I_2
I_3
I_4
I_5
I_6
I_7
I_8
I_9
T_0
T_1
S_2
S_3
S_4
S_5
S_6
S_7
RS_0
RS_1
R_1
R_2
R_3
R_4
R_5
R_6
R_7
R_9
R_{10}
R_{11}
R_{12}
R_{13}
0

2 Searching for a table of period 32

For the outer loop, the doubling is borrowed with tweaking from [1]. The constants are produced by machines which were discovered by automatic (although not exhaustive) search. There is some micro-optimisation, so that instead of pushing a unary 8 and then doubling, the machine only pushes 7 and then steps into the doubling machine partway through its execution.



3 Possible optimisations

Although the searches for small machines for the components of the outer loop were not exhaustive, they covered a fair amount of the search space. There are more likely to be opportunities for saving one or maybe two states in each of the components of the inner loop, and possibly even more by working with the machine as a whole rather than designing individual components and then linking them together.

4 Conclusion

The smallest two-symbol Turing machine whose termination is *unknown* has only 5 states. The smallest known two-symbol TM whose termination

is *unknowable* in ZFC has about 1900 states. We have presented a 64-state machine which occupies a kind of middle ground: it is known to terminate if we assume a much stronger axiom than ZFC; but it is not known whether its termination is knowable in ZFC.

5 Appendix: basic properties of Laver tables

Taking as our definition of Laver tables the two properties $x \star_n 1 = x + 1$ (working always $\bmod 2^n$, although for ease of presentation we will omit this henceforth) and $x \star_n (y \star_n z) = (x \star_n y) \star_n (x \star_n z)$ we can derive the recurrence used by observing that $y = (y-1) \star_n 1$, so that $x \star_n y = x \star_n ((y-1) \star_n 1) = (x \star_n (y-1)) \star_n (x \star_n 1) = (x \star_n (y-1)) \star_n (x+1)$.

Let $N = 2^n$. We can prove $N \star_n y = y$ by induction: $N \star_n 1 = (N+1) = 1$ gives the base case, and $N \star_n (y+1) = (N \star_n y) \star_n (N \star_n 1) = y \star_n 1 = y+1$ gives the inductive step.

Then if $x < N$ we can show that $x \star_n y > x$ by a nested induction. The outer induction ranges over x from $N-1$ down to 1, and the inner one over y from 1 to N. The base case for the outer induction is that $(N-1) \star_n y = N$: by definition, $(N-1) \star_n 1 = N > N-1$; and then $(N-1) \star_n (y+1) = ((N-1) \star_n y) \star_n N = N \star_n N = N$. The inductive step for the outer induction descends over x. By definition $x \star_n 1 = x + 1 > x$ (since $x < N$). The inner inductive step splits into two cases: $x \star_n (y+1) = (x \star_n y) \star_n (x \star_n 1)$, and by the inner inductive hypothesis $x \star_n y > x$; either $x \star_n y < N$ and we apply the outer inductive hypothesis to conclude that $(x \star_n y) \star_n (x \star_n 1) > x \star_n y > x$; or $x \star_n y = N$ and $(x \star_n y) \star_n (x \star_n 1) = N \star_n (x \star_n 1) = x \star_n 1 > x$.

Note that this proof also constitutes a proof that the recurrence eventually terminates, and is an effective (if not an efficient) means of calculating Laver tables.

Since the periodicity of $1 \star_n y$ is a key element of this paper, we also prove that every row of the table except $N \star_n y$ is periodic with a period less than N. Firstly note that if $x \star_n a = x \star_n b$ then $x \star_n (a+1) = (x \star_n a) \star_n (x+1) = (x \star_n b) \star_n (x+1) = x \star_n (b+1)$, so that if any value occurs more than once in the row we have periodicity. But since for $x < N$ we have $x \star_n y > x$, the only row which can have N distinct values is $N \star_n y$: by the pigeonhole principle, every other row must have repeated values. And since we're working modulo N and the repetition wraps around from $x * N$ to $x * 1$, the period must be a factor of N, and hence a power of two.

Finally, we observe that the periodic cycle of every row, which starts with $x \star_n 1 = x + 1$, ends with $x \star_n y = N$ for some y. In particular, $x \star_n N = N$. Proof: we've already shown that $N \star_n N = N$. But since for all $x < N$ we have $x \star_n y > x$, if $z \star_n z = z$ then z must be N. Now, $x \star_n N = x \star_n (N \star_n N) = (x \star_n N) \star_n (x \star_n N)$, so $x \star_n N = N$.

References

[1] "Deedlit11" (2013) Okay, more Turing machines. http://googology.wikia.com/wiki/User_blog:Deedlit11/Okay,_more_Turing_machines, retrieved 2016-05-28.

[2] Laver (1995) On the algebra of elementary embeddings of a rank into itself. *Advances in mathematics*, 110(2), 334-346.

[3] O'Rear (2016) NQL, https://github.com/sorear/metamath-turing-machines, retrieved 2016-05-28.

[4] Yedidia and Aaronson (2016) A relatively small Turing machine whose behavior is independent of set theory. Preprint at arXiv:1605.04343 [cs.FL]