Improved Bounds for Functions Related to Busy Beavers

Authors' names and addresses:

Amir M. Ben-Amram (contact author)
The Academic College of Tel-Aviv-Yaffo,
4 Antokolsky St.,
Tel-Aviv 64044, Israel.
email: amirben@mta.ac.il

Holger Petersen
The University of Stuttgart,
Institute of Computer Science, Breitwiesenstr. 20–22,
D-70565 Stuttgart, Germany.
email: petersen@informatik.uni-stuttgart.de

Improved Bounds for Functions Related to Busy Beavers

Amir M. Ben-Amram* Holger Petersen†

November 13, 2001

Abstract

Consider Turing machines that use a tape infinite in both directions, with the tape alphabet $\{0, 1\}$. Rado's *busy beaver* function $\text{ones}(n)$ is the maximum number of 1s such a machine, with n states, started on a blank (all-zero) tape, may leave on its tape when it halts. The function $\text{ones}(n)$ is noncomputable; in fact, it grows faster than any computable function.

Other functions with a similar nature can also be defined. All involve machines of n states, started on a blank tape. The function $\text{time}(n)$ is the maximum number of moves such a machine may make before halting. The function $\text{num}(n)$ is the largest number of 1s such a machine may leave on its tape in the form of a single run; and the function $\text{space}(n)$ is the maximum number of tape squares such a machine may scan before it halts.

This paper establishes new bounds on these functions in terms of each other. Specifically, we bound $\text{time}(n)$ by $\text{num}(n + o(n))$, improving on the previously known bound $\text{num}(3n + 6)$. This result is obtained using a kind of "self interpreting" Turing machines. We also improve on the trivial relation $\text{space}(n) \leq \text{time}(n)$, using a technique of counting crossing sequences.

1 Introduction

The Busy Beaver Game is based on deterministic Turing machines with one bi-infinite work-tape. In each step a machine reads a tape symbol from the set $\{0, 1\}$, writes a (possibly identical) symbol, shifts its head one square to the left or to the right, and enters a new state. There is a single halt state, and on its transition to this state the machine can still write a symbol.

*The Academic College of Tel-Aviv-Yaffo, 4 Antokolsky St., Tel-Aviv 64044, Israel.

†The University of Stuttgart, Institute of Computer Science, Breitwiesenstr. 20–22, D-70565 Stuttgart, Germany. This paper was written while the author was visiting the Academic College of Tel-Aviv-Yaffo. Support by "Deutsche Akademie der Naturforscher Leopoldina", grant number BMBF-LPD 9901/8-1 of "Bundesministerium für Bildung und Forschung" is gratefully acknowledged.

The game proposed by Rado [14] is to determine values of the function ones(n). Here ones(n) is defined to be the maximum number of 1's a machine with n states (not counting the halt state), that halts when started on a blank (all zero) tape, can leave on its tape when halting. An n-state machine producing that many 1's is called a Busy Beaver (note that there might be several Busy Beavers with the same number of states). Rado proved that the function ones(n) grows faster than every computable function. The same is true for time(n), the maximum number of steps executed by a machine with n states (this is also called the shift function), and for space(n), the maximum number of tape cells visited by machines of this kind. For space(n) only those cells count that are visited by states other that the halt state of the machine. If $n \geq 2$ the transition to the halt state can always be redirected to a cell previously visited (this is of course impossible for 1-state machines). For the definition of Busy Beavers, the 1's may be left on the tape in any arbitrary arrangement. The function num(n) is defined the same as ones(n), except that the 1's left are required to form a single run; thus only part of the n-state machines qualify as "num contestants".

A machine which attains the maximum number of 1's (respectively, number of moves, cells visited, 1's in a run) is called a ones- (respectively time-, space-, num-) champion. A ones-champion is of course a Busy Beaver; a time champion is sometimes referred to as a shift champion.

Although (or perhaps because) these functions are non-computable in general, many investigators, including Rado himself, have tried to determine values for small n. As pointed out by Chaitin [4], open problems such as the Goldbach Conjecture can be connected to the computation of ones(n) for a specific n, and thus the chances of extending our knowledge beyond the first few values are slim.

Traditionally, a Turing machine M participating in the Busy Beaver competition is represented by a table of the form

$$
\begin{array}{rc|c|c|}
 & & \multicolumn{2}{c}{\text{input symbol}} \\
 & & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{1} \\
\cline{3-4}
\text{current state} & 1 & w_1^0 m_1^0 s_1^0 & w_1^1 m_1^1 s_1^1 \\
 & 2 & w_2^0 m_2^0 s_2^0 & w_2^1 m_2^1 s_2^1 \\
 & \vdots & \vdots & \vdots \\
 & n & w_n^0 m_n^0 s_n^0 & w_n^1 m_n^1 s_n^1 \\
\cline{3-4}
\end{array}
$$

where $w_k^i \in \{0,1\}$ indicates the symbol written by M after reading i in state k, $m_k^i \in \{L, R\}$ is the direction of the head movement, and $s_k^i \in \{1, \ldots, n+1\}$ the new state M enters. State 1 is the initial state and state $n+1$ is the halt state.

As an example, here is a Busy Beaver for $n = 2$:

		input symbol	
		0	1
current state	1	$1R2$	$1L2$
	2	$1L1$	$1R3$

The progress made in the chase for Busy Beavers and other champions is reflected by the following table, where the values for $n = 5$ and $n = 6$ are lower bounds:

n	ones(n)	num(n)	space(n)	time(n)	references
1	1	1	1	1	[14]
2	4	4	4	6	[14]
3	6	6	7	21	[9, 1]
4	13	12	16	107	[16, 2, 1]
5	$\geq$4098	?	$\geq$12289	$\geq$47176870	[11]
6	$\geq 2.5 \cdot 10^{21}$	?	?	$\geq 5 \cdot 10^{42}$	[10]

The lower bound on space(5) was derived from Michel's analysis of a ones-contender establishing ones(5) $\geq$ 4098 [12].

A recursive lower bound on num(n) has been given by Green [5]. He proved num(n) $\geq G_n(0)$, where $G_n(i)$ is a function that satisfies

$$A(n,n) < G_{4n+3}(0) < A(4, 2n+1)\,.$$

Here A is the Ackermann-function defined as in [7]:

$$\begin{aligned} A(0,y) &= 1 \\ A(1,0) &= 2 \\ A(x,0) &= x+2 \qquad (x \geq 2) \\ A(x+1,y+1) &= A(A(x,y+1),y)\,. \end{aligned}$$

It is well known that such a function grows faster than every primitive recursive function.

Another goal, which appears not to be as hopeless as the actual computation of the functions, is to find relations between ones(n), time(n), space(n) and num(n). It is trivial that

$$\mathrm{num}(n) \leq \mathrm{ones}(n) \leq \mathrm{space}(n) \leq \mathrm{time}(n), \tag{1}$$

and Rado [14, 15] found that

$$\mathrm{time}(n) \leq (n+1) \times \mathrm{ones}(5n) \times 2^{\mathrm{ones}(5n)}.$$

Buro [3, pp. 5–6] gave a construction showing time(n) $\leq$ ones($9n$). Ben-Amram, Julstrom and Zwick [1] gave several new bounds. We cite

Theorem 2 (of [1]) $\mathrm{time}(n) \le (2n-1) \times \mathrm{ones}(3n+3)$.

Theorem 7 (of [1]) $\mathrm{time}(n) < \mathrm{num}(3n+6)$.

Ben-Amram, Julstrom and Zwick point out that conceivably, the upper bound of their Theorem 2 is much better asymptotically than that of Theorem 7; but the known inequalities between $\mathrm{ones}(n)$ and $\mathrm{num}(n)$ do not suffice for proving that. There was similarly no proof for the intuitive claim, based on the astounding growth rate of $\mathrm{ones}(n)$, that Buro's bound is almost always worse than Rado's. However, Theorem 7 was a clear improvement on Rado's bound as well as on Buro's, since $\mathrm{num}(3n+6) \le \mathrm{ones}(3n+6) \ll \mathrm{ones}(5n)$. To sum up, the goal in this line of research is to make the argument of the bounding function as close to n as possible, with the best results so far being around $3n$.

For appreciating the novelty in the proof technique we use, let us sketch the way in which the results quoted from [1] were obtained. The heart of the technique is transforming a given n-state machine M into a machine M', instrumented for profiling the computation. More precisely, M' simulates M on alternate tape squares, say the odd-numbered squares, while the remaining (even-numbered) squares are used for profiling. In the simplest construction, these squares are marked with 1's leading to a machine whose ones production surpasses $\mathrm{space}(n)$. To obtain a bound on $\mathrm{time}(n)$ in terms of $\mathrm{ones}()$, the simulator counts steps executed by M, again using the profiling squares to hold the count. In the part of M' which simulates M, every transition of M is replaced with two transitions in the same direction, thus skipping over the profiling squares; to this end a new intermediary state is inserted. Assuming that M has $2n-1$ non-halting transitions, the number of states in M' becomes at least $3n-1$. We do not see a way of reducing the number below $3n$ within this approach. However by using *interpretation* instead of *instrumentation*, we will be able to break the $3n$ barrier.

Our construction is motivated by the following wishful thought. Suppose a Turing machine could inspect its own finite control, like a stored program on a von-Neumann machine. Then we could construct a machine by attaching an n-state champion M to a fixed interpreter; the interpreter inspects M's instructions and simulates them while counting steps, obtaining when M halts a unary count of M's steps. This would give a machine M' whose size is n plus a universal constant c_{time}, yielding the result

$$\mathrm{time}(n) \le \mathrm{num}(n + c_{\mathrm{time}})$$

a result involving an additive constant only in the right-hand side, in contrast with the multiplicative constant of 3 in the previous results (in other words, this reduces the multiplicative factor in the upper bound from 3 to 1). Such a result would imply interrelations among the four of $\mathrm{num}(n)$, $\mathrm{ones}(n)$, $\mathrm{time}(n)$, and $\mathrm{space}(n)$ involving additive constants only, obtaining major progress towards proving the conjectures in [1]. Unfortunately, a Turing machine *cannot* inspect its finite control, and thus we do not know if the multiplicative factor can be reduced to one. By using a less efficient method of extracting information from the finite control we do manage to reduce the factor to below 2, i.e., we bound time(n) by num($n + o(n)$). For the precise result see Theorem 1 in Section 4. Theorem 2 in that section is an improvement on the trivial relation space(n) $\leq$ time(n), obtained by a technique of counting crossing sequences.

2 Compressing Champions

The encoding of arbitrary n-state Turing machine tables as introduced in Section 1 seems to require $2n\lceil\log_2 n\rceil$ bits for state transitions and $\Theta(n)$ further bits for specifying head movements, symbols written, and the occurrence of the halt state. It is however easy to see, that for the purpose of describing a *champion* certain portions of the table can be fixed, e.g., the first transition can be assumed to move the head right by noticing that timing, space usage etc. do not change if left and right movements are interchanged consistently. Another remark of this kind is that a champion can be assumed to use all of its states.

More significant is the following:

Observation 1 *For each n and each of the measures there is a champion M, such that in its computation on a blank tape "new" states (states not previously reached) appear in increasing order, i.e., the first state visited and not in the set $\{1, \ldots, k\}$ is $k+1$.*

The observation is seen to be valid by renaming the states appearing in the unique computation of M on a blank tape. Reductions of the number of machines to be considered, based on similar normalizations, have been used in computing or approximating ones(n) for small n, see [11].

The transition that is followed when a state k is first arrived at will be called *special*. All other transitions are *ordinary*. It is now easy to observe, that targets of special transitions can be omitted from a description of M, as long as there is an indication of which transition is special. By keeping track, during execution, of the states already visited, the target of the special transition can be established. Note also that the number of special transitions is exactly n, since the halt state has to be reached. This also characterizes the transition leading to the halt state.

We conclude that M can be described by the following information:

1. The number $n-1$ in self-delimiting binary notation, using at most $2\lceil\log_2 n\rceil$ bits (more compact notations exist, but the saving will not make a substantial difference in our results).

2. An array containing 6 bits for every state $i \in \{1, \ldots, n\}$. These bits correspond to the six components of every row of the transition table, where the last component of each triple (new state) is replaced by a single bit that is 1 if and only if the transition is special.

3. A list of the n destinations of ordinary transitions. The list is sorted according to their first appearance in M's computation on a blank tape. Each destination can be encoded in $\lceil\log_2 n\rceil$ bits.

3 Simulating a ROM

Our aim is to include within a Turing machine's finite control a ROM of m bits. The machine should be able to dump its contents to the tape during an initialization phase. A trivial implementation is having m states assigned to this task, each state responsible for generating a single bit. We will now show that a Turing machine can use a group of m states for encoding information more efficiently and retrieve about $m \log m$ bits. By retrieving we mean that a fixed main program plus these m states can reconstruct the contents of the ROM.

Lemma 1 *A group of $m \geq 2$ states can encode a string of $\lfloor m \log_2 m \rfloor$ bits.*

Proof. Consider the sequence of bits to be stored as a binary number $k < m^m$. Let $k = \sum_{i=0}^{m-1} k_i m^i$, where $0 \leq k_i \leq m-1$ for $0 \leq i \leq m-1$ (the m-ary representation of k). We will use m states as a subroutine, that is entered at state b (the base), consists of states $b+i$, $0 \leq i \leq m-1$ and returns control to the main program by entering a specific state $e \notin [b, b+m-1]$.

The transition table for the m states is as follows:

	0	1
$b+i$, $0 \leq i < m-1$	$0R(b+i+1)$	$1R(b+m-1-k_i)$
$b+m-1$	$0Re$	$1R(b+m-1-k_{m-1})$

The main program extracts the ROM contents in the following way. First it sets up an area of squares that is marked on the left by '11', followed by infinitely many 0's (short notation: 110^ω). Naturally, this area will be to the right of the portion of the tape used so far by the computation. The machine now enters state b while moving into the first square of the 0^ω. From the transition table shown above it is easy to see that the machine then moves m squares to the right before the subroutine "returns" to state e. Now, the machine moves to the left, counting squares until it hits the 1. Thus the value of m has been obtained. The machine proceeds to extract the values k_i for $0 \leq i \leq m-1$. The procedure is similar: to extract k_i, the machine sets up an area containing $10^{i+1}10^\omega$, and enters state b while moving into the third square of this string (after the initial 10). The computation from this point proceeds through the following states (moving ever to the right):

$$b \to \cdots \to (b+i) \to (b+m-1-k_i) \to \cdots \to (b+m-1) \to e$$

Thus when control returns to state e, the head's position is $k_i + 2$ squares to the right of the single 1. By moving backwards and counting, the machine obtains the value of k_i. Note that despite the fact that the subroutine always returns to state e the main program can figure out that it has now decoded k_i by counting the zeros between the two initial 1's. Hence, the program can manage the progression from i to $i+1$ until all the k_i values have been extracted. $\square$

4 Results

4.1 An improved bound on $\mathrm{time}(n)$ in terms of $\mathrm{num}()$

We combine the techniques introduced in the preceding sections for proving a relation between $\mathrm{time}(n)$ and $\mathrm{num}(n)$.

Theorem 1 *There is a constant c_{time}, such that*

$$\mathrm{time}(n) \leq \mathrm{num}(n + \lceil 8n/\log_2 n\rceil + c_{\mathrm{time}})$$

for all $n \geq 2$.

Proof. Consider an n-state machine M executing $\mathrm{time}(n)$ steps on an empty input tape (a time champion). By the discussion of Section 2, M can be described in $c(M)$ bits, where

$$c(M) = 2\lceil\log_2 n\rceil + 6n + n\lceil\log_2 n\rceil \leq 2\log_2 n + 7n + n\log_2 n + 2.$$

Let $m = n + \lceil 8n/\log_2 n\rceil$ and observe that for $n \geq 2$ we have

$$2\log_2 n + 7n + n\log_2 n + 2 \leq m\log_2 m\,.$$

By Section 3, a description of M can be encoded using m states.

Consider now a machine N that first writes the description of M on its tape and then simulates the computation of M on a blank tape. Since the description of M is "compressed" according to Section 2, the simulation starts with a partially specified transition table, in which all destinations of transitions are still unknown. In the course of the simulation N fills in the missing portions by inspecting the description of M whenever a transition is reached for the first time. During the simulation, N keeps track of the number of simulated steps, in the form of a unary number (run of 1's); on completion, N erases everything but this number, to qualify as a "num" contestant. This proves the result, as the size of N is bounded by m (for the ROM) plus a constant (the rest of N is a fixed program). Just choose c_{time} at least as large as that constant. $\square$

Corollary 1 *For any computable function f, a constant d_f exists such that*

$$\mathrm{ones}(n + \lceil 8n/\log_2 n\rceil + d_f) \geq f(\mathrm{ones}(n))$$

for all n. The relation also holds with $\mathrm{space}()$ *and* $\mathrm{time}()$ *substituted for* $\mathrm{ones}()$.

Proof. Without loss of generality we assume f to be increasing. Since f is computable, a num-contestant can be augmented with a fixed subroutine that applies f to the unary number generated by the contestant. Let c_f be the number of states in this subroutine and let $d_f = c_f + c_{\text{time}}$. Then

$$\text{num}(n + \lceil 8n/\log_2 n \rceil + d_f) \geq f(\text{num}(n + \lceil 8n/\log_2 n \rceil + c_{\text{time}})) \geq f(\text{time}(n))$$

and applying trivial inequalities (1) on both sides gives the result. $\square$

Remark. We have not included num() in the corllary, because for this function a similar but stronger result is known, namely: for any computable function f, a constant c_f exists such that

$$\text{num}(n + c_f) \geq f(\text{num}(n))$$

(this is Theorem 9 of [1]).

As a peculiar consequence, we are now able to tell that Buro's upper bound ones$(9n)$ on time(n) is asymptotically worse than Rado's $(n+1) \times \text{ones}(5n) \times 2^{\text{ones}(5n)}$. To see this, let $f(x) = x^2 \cdot 2^x$. The corollary shows that for sufficiently large n we have

$$f(\text{ones}(5n)) \leq \text{ones}(6n) \ll \text{ones}(9n)\,.$$

4.2 An improved bound on space(n) in terms of time(n)

Next we use the technique of counting crossing-sequences known from complexity theory (e.g., [6, 7]) for improving on the trivial relationship space$(n) \leq$ time(n). In order to simplify the counting we don't prove the strongest bound possible as far as constant factors are concerned.

Theorem 2 *For $n \geq 2$:*

$$\text{time}(n) \geq \left(\frac{3}{4} + \frac{\log_2 \text{space}(n)}{4 \log_2 n}\right) \times \text{space}(n),$$

where the first factor is greater than 1 and in general grows faster than any recursive function.

Proof. For $2 \leq n \leq 4$ the relation can be checked directly; for the rest of the proof assume $n \geq 5$. Consider the computation of an n-state machine M using space(n) tape cells (a space champion) on a blank tape. Since a cell that is reached for the first time by the halting transition does not count, in what follows we neglect the transition of M into the halt state. The crossing sequence (c.s.) at each border between two tape cells is the sequence of states the machine enters each time it passes the respective border. There are space$(n) - 1$ non-empty c.s.'s. The starting position and the final position of M's head partition the tape into three regions. The left and right regions contain c.s.'s of even length, while the central region contains c.s.'s of odd length. We now omit all c.s.'s of the shorter of the left or right region (or the left region, if they are of equal length). Observe that this leaves at least $\frac{1}{2} \times \text{space}(n) - 1$ crossing sequences. No two of these sequences can be identical, because identical c.s.'s are from the same region (parity), thus it would be possible to collapse non-empty portions of the computation to obtain another valid computation, thus contradicting the determinism of M. Let

$$\ell = \frac{\log_2 \text{space}(n)}{\log_2 n} - 1.$$

The number of different possible c.s.'s of length between 1 and ℓ is

$$\frac{n^{\ell+1} - 1}{n - 1} - 1 = \frac{n^{\log_2 \text{space}(n)/\log_2 n} - 1}{n - 1} - 1 = \frac{\text{space}(n) - 1}{n - 1} - 1.$$

Since $n \geq 5$ we have

$$\frac{\text{space}(n) - 1}{n - 1} - 1 \leq \frac{1}{4} \times \text{space}(n) - 1,$$

and therefore at least $\frac{1}{4} \times \text{space}(n)$ of the c.s.'s have a length exceeding ℓ. Since each entry in a c.s. corresponds to a step, this accounts for at least

$$\frac{1}{4} \times \text{space}(n)(\ell + 1) = \frac{\text{space}(n) \times \log_2 \text{space}(n)}{4 \log_2 n} \tag{2}$$

steps. To this count we can add $\frac{3}{4} \times \text{space}(n) - 1$ c.s.'s which contain at least one step each; and finally the single halting transition, obtaining

$$\text{time}(n) \geq \left(\frac{3}{4} + \frac{\log_2 \text{space}(n)}{4 \log_2 n} \right) \times \text{space}(n)\,.$$

The first factor is greater than 1 because for $n \geq 2$, $\text{space}(n) > n$. Its non-recursive growth rate follows immediately from the corresponding property of $\text{space}(n)$. □

5 Concluding Remarks

In this paper we have improved bounds relating functions arising from the Busy Beaver Game. Recently some work has been done on a variant, the 4-tuple version of the problem [13]. Here each transition can either write a symbol *or* move the tape-head, but not perform both actions. Additionally, this version of the game requires the 1's produced to occur in a single run (as in the definition of num) and the head has to be positioned, on halting, on the leftmost 1. Our construction in Section 3 can be carried out with 4-tuple machines; therefore, with an adjusted additive constant, a result analogous to Theorem 1 holds also for this variant.

We remark, that a 4-tuple machine can be converted into a 5-tuple version, as in the standard Busy Beaver game, without increasing the number of states (except that on halting, the head may be left near rather than on the leftmost 1). It follows that the results of the 4-tuple game are lower bounds on the values of num() in the standard version. Thus the 4-tuple score function is lower than the four functions we considered: this makes the discovery of actual values for small n somewhat more tractable in this version.

Promising 4-tuple candidates for state numbers greater than 5 have been found by genetic algorithms, see [13]. The standard, 5-tuple version had served as a test-case in this area of research earlier [8]. It thus seems that the Busy Beaver is becoming the guinea-pig of genetic algorithms.

Another observation is that the technique from Section 3 can be used to obtain a lower bound on the number of inequivalent Turing machines of n states. This is because for large n, a fixed 'extractor' can generate approximately n^n different results on an initially blank tape. The number of inequivalent machines is of importance, since previous attacks on Busy Beaver problems were mainly based on a reduction of the number of machines to be inspected by making use of equivalence classes.

References

[1] A. M. Ben-Amram, B. A. Julstrom, and U. Zwick. A note on Busy Beavers and other creatures. *Mathematical Systems Theory*, 29:375–386, 1996.

[2] A. H. Brady. Solution of the non-computable "Busy Beaver" game for $k = 4$. In *ACM Computer Science Conference (Washington DC, 1975)*, page 27, 1975.

[3] M. Buro. Ein Beitrag zur Bestimmung von Rados $\Sigma(5)$ oder Wie fängt man fleißige Biber? Technical Report 146, Rheinisch-Westfälische Technische Hochschule Aachen, 1990.

[4] G. J. Chaitin. Computing the Busy Beaver function. In *Information, Randomness & Incompleteness: Papers on Algorithmic Information Theory, Gregory J. Chaitin, World Scientific, Series in Computer Science–Vol. 8*, pages 74–76. 1987.

[5] M. W. Green. A lower bound on Rado's Sigma Function for binary Turing machines. In *Switching Circuit Theory and Logical Design, Proceedings of the Fifth Annual Symposium (Princeton, N.J., 1964)*, pages 91–94. The Institute of Electrical and Electronics Engineers, Inc., 1964.

[6] J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley, Reading, Mass., 1969.

[7] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass., 1979.

[8] T. Jones. Crossover, macromutation, and population-based search. Technical Report WP 95-02-24, Santa Fe Institute, 1995.

[9] S. Lin and T. Rado. Computer studies of Turing machine problems. *Journal of the Association for Computing Machinery*, 12:196–212, 1965.

[10] H. Marxen and J. Buntrock. List of record machines. Available at: `http://www.drb.insel.de/~heiner/BB/bb-list`.

[11] H. Marxen and J. Buntrock. Attacking the Busy Beaver 5. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 40:274–251, 1990.

[12] P. Michel. Busy beaver competition and Collatz-like problems. *Arch. Math. Logic*, 32:351–367, 1993.

[13] F. B. Pereira, P. Machado, E. Costa, and A. Cardoso. Graph based crossover – A case study with the Busy Beaver problem. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1149–1155, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[14] T. Rado. On non-computable functions. *The Bell System Technical Journal*, 41:877–884, 1962.

[15] T. Rado. On a simple source for non-computable functions. In *Proceedings of the Symposium on Mathematical Theory of Automata (New York, 1962)*, volume XII of *Microwave Research Institute Symposia Series*, pages 75–81. Polytechnic Press of the Polytechnic Institute of Brooklyn, 1963.

[16] B. Weimann, K. Casper, and W. Fenzl. Untersuchungen über haltende Programme für Turingmaschinen mit 2 Zeichen und bis zu 5 Befehlen. In *GI Gesellschaft für Informatik e.V.: 2. Jahrestagung (Karlsruhe, Germany, 1972)*, Lecture Notes in Economics and Mathematical Systems, pages 72–81, Berlin-Heidelberg-New York, 1973. Springer.