Jochen Ludewig, Uwe Schult, Frank Wankmüller

Chasing the Busy-Beaver - Notes and Observations on a Competition to Find the 5-state Busy Beaver

Bericht Nr. 159, 1983

Abstract

This is a report on the results of a competition which was initiated on the occasion of the 6th GI-conference on Theoretical Computer Science, which took place at the University of Dortmund from January 5th to 7th, 1983. It was asked for the best solution of the 5-state Busy-Beaver-Game.

At first we make some historical remarks, introduce the formalism, and list some results. Then the two best solutions are described. Next we make some remarks on the behaviour of good beavers and on the strange behaviour of some Turing machines. Zoological names were given to the latter machines. The amusing results are written down in the last chapter. In the appendix you can find a lot of examples.

Adresses

Jochen Ludewig
Brown Bovery Research Center
CH 5405 Baden
Switzerland

Uwe Schult
Vorlandring 15
D 2000 Hamburg 74
West Germany

Frank Wankmüller
Informatik II
Universität Dortmund
Postfach 500 500
D 4600 Dortmund 50
West Germany

0. Contents

1. Some Historical Remarks

In 1936 Turing introduced Turing machines for the first time as a formulation of a concept of algorithm [1]. A more precise definition was developped by Kleene in 1952 [2]. 10 years later Rado restricted Turing machines to machines with only two different tape symbols, namly ƀ ("blank") and | ("bar"), which have to move their read-write-head in each step during the computation [3,4]. Main topics of these investigations are the function S(n) and Σ(n). S(n) is defined to be the maximum number steps, which can be done by such a Turing machine with n states which starts its computation with the empty tape and stops after a finite number of steps. Σ(n) is defined to be the maximum number of bars, which can be produced by such an n-state Turing machine, which starts its computation with the empty tape and stops after a finite number of steps. Obviously, S(n) and Σ(n) are not recursive.

Rado formulated the problem of computing Σ as a game, which is called the Busy-Beaver-Game. For a given n one has to construct a Turing machine with n states and the two tape symbols ƀ and |. The winner is the lucky person, whose machine stops, if the computations is started with the empty tape and whose machine produced more bars than the other ones. The Busy-Beaver-Game can truly be renamed the "Busy Beaver Disease" for it has some properties of an infection. Once infected the healing is very difficult and takes an enormous amount of time (especially CPU-time of a computer; see also chapter 3 and 4 of this report). Some cases are even told to be chronic. Nevertheless, the Busy-Beaver-Game stimulated results in theory of computation, and in spite of the fact that Σ is not computable, uncounted computer scientists and amateurs tried to find the values of Σ for small n. Their experience gives us a strong hint to the enormous speed, with which the complexity of combinatorial problems increases.

For n=1 or n=2 the solution of the Busy-Beaver-Game is trivial. The first successful computation of $\Sigma(3)$ was done by Lin in 1963 [5]. By some equivalence relations Lin was able to reduce the number of machines to be checked from 16,777,216 to 82,944. Then he checked all these 82,944 machines by a computer. In 82,904 cases the program was able to decide whether the machine stopped or not. The remaining 40 machines had to be checked by had. So, Lin proved $\Sigma(3)=6$.

s	r	s'	w	m
1	ƀ	2	|	R
1	|	3	|	L
2	ƀ	3	|	R
2	|	1	|	S
3	ƀ	1	|	L
3	|	2	ƀ	L

Fig.1: A 3-state TM, which stops after 11 steps and produces 6 bars, if the computation is started with the empty tape. The initial state is 1.

10 years later, Weimann et al. gave some better equivalence relations and better computer programs to solve $\Sigma(4)$ [6]. If one would apply the method of Lin, 100,663,296 machines must be tested. 15,000 would be left to be checked by hand. Weimann reduced the number of machines from 25,600,000,000 to 1,198,690. 396 are to be checked by hand. Weimann discovered two machines producing 13 bars:

s	r	s'	w	m
1	ß	2	|	R
1	|	3	ß	R
2	ß	1	|	L
2	|	1	|	R
3	ß	1	|	S
3	|	4	|	R
4	ß	4	|	L
4	|	2	ß	L

(a)

s	r	s'	w	m
1	:	2	|	R
1	¡	2	|	L
2	ß	1	|	L
2	|	3	ß	L
3	ß	1	|	S
3	|	4	|	L
4	ß	4	|	R
4	|	1	ß	R

(b)

Fig. 2: Two 4-state TM, which stop after 96 respectivly 107 steps and produce 13 bars, if the computation is startet in state 1 with the empty tape.

In fact, these two solutions are very similar. The only difference is the initial state. If one chooses 2 as initial state of machine (a) respectibly (b), the machine will behave like the other one.

In 1973 Weimann gave an example of a 5-state TM, which produces 40 bars [7]. This result improved the best known lower bound for $\Sigma(5)$ more than two times.

On the occasion of the 6th GI-Conference on Theoretical Computer Science [9], which took place at the University of Dortmund from January 5th to 7th 1983, a competition was initiated for the best solution of the 5-state Busy-Beaver-Game for three reasons:

- to stimulate further investigations of the problem
- to obtain better examples for th complexity of the problems involved
- or simply to encourage the desire to play.

In this report we summarize the results of the competion. After this short survey we give a short introduction to the formalism and list some results.

In chapter 3, Mr. Schult, the winner of the competition, describes how he attacked the problem.

Chapter 4 is the report of the runner-up Mr. Ludewig.

Chapter 5 contains some remarks on the behaviour of good beavers. This chapter is also due to Mr. Ludewig.

During chasing the busy beaver Mr. Ludewig discovered some Turing machines with strange behaviour. According to that Mr. Ludewig gave zoological names to them. This amusing results are written down in chapter 6.

Finally, we give some examples and statistics in the appendix.

I have to thank all the busy hunters, who spent a lot of time in the competition and made it such successful. Furthermore, I have to thank Mrs. U. Minning for the excellent typing of the script. And last, not least, I should mention, that without help of Prof. Dr. V. Claus it would have been impossible to carry out the competition.

2. A Short Introduction to the Formalism

Turing machines are well-known to be 5-tupels $(X, S, \delta, \text{ƀ}, s_o)$ with:

(i) X is an alphabet, which contains the so-called "blank symbol" ƀ.

(ii) S is a finite set of states, which contains the so-called initial state s_o.

(iii) $\delta: S \times X \to S \times X \times \{R, L, S\}$ is the transition function. $(s,x) \longmapsto (s', y, \begin{Bmatrix} R \\ L \\ S \end{Bmatrix})$ means, that, if the machines is in state s and its read-write-head reads an x on the tape, the state is changed to s', x is repleced by y, and the read-write-head is moved either to the right (R), or to the left (L), or the computation stops (S).

Busy-Beaver-Turing-machines are a special kind of Turing-machines. A n-Busy-Beaver-Turing-machine (abbreviation: n-BBT) is a Turing machine with $X = \{\text{ƀ}, |\}$, $S = \{1,2,\ldots,n\}$, $s_o = 1$, and the computation stops after a finite number of steps, if the machine has as input the empty tape, i.e. all squares of the tape contain the blank symbol. | is called "bar symbol". Now, we can define following functions:

$\Sigma(n)$ = maximum number of bars, which can be produced by an n-BBT, which starts with the empty tape.

$S(n)$ = maximum number of steps, which can be done by an n-BBT, which starts with the empty tape.

In the literature, Σ is often called "Rado's Σ - function". The Busy-Beaver-problem consists of computing $\Sigma(n)$ for given n.

The following two properties of Σ can be easily shown:

1. $\Sigma: \mathbb{N} \to \mathbb{N}$ is not recursive.
2. For each recursive function $f: \mathbb{N} \to \mathbb{N}$ there is an $n_o \in \mathbb{N}$ such that for all $n \geq n_o$ $f(n) \leq \Sigma(n)$ holds

Up to now, the following results are known [7,8]:

n	$\Sigma(n)$	S(n)
1	1	1
2	4	6
3	6	21
4	13	107
5	≥40	≥992
6	≥40	
7	≥22.961	
8	$\geq 3 * (7,3^{92}-1)/2$	

Fig. 3

It seems to be very difficult, to solve the Busy-Beaver-Problem for n larger than 4, because of the enormous number of different Turing-machines with n states and the alphabet {b,|} (see the following table):

n	number of possible Turing machines
1	64
2	20,736
3	16,777,216
4	25,600,000,000
5	63,403,380,965,376
6	232,218,265,089,212,416

Fig. 4

During the competition 133 Turing machines were sent to Dortmund. It took more than 4,000 CPU-seconds on a SIEMENS 7.748-computer to simulate them and to verify the results. Out of all the machines submitted by each participant we selected the best and obtained the following list of winners.

place	name	bars	steps
10	Bruno Weimann (1973)	40	556
8	Gert Lormes & M. Bohlen	47	653
8	Jürgen Christoffel & Co.	47	653
7	Hans Zschintsch	68	3,737
6	Fritz Brinkmeier	71	3,902
5	Wolfgang Hatzel	87	3,190
4	Dietmar Hehmann	163	14,975
3	Kl. Muuss & H. Sönnichsen	168	21,294
2	Jochen Ludewig	240	41,360
1	Uwe Schult	501	134,467

Fig. 5

3. Chasing the Busy-Beaver, Part I

3.1 General Attack

A basic method is the enumeration of all Turing programs. Given five states (s) and two symbos (x), the Turing program has 10 entries. Each entry may hold five different next states (s'), two different symbols (y) to write, and three different head-moving instructions (m), namely L, R, S given 30 possible entries. If m = S, the next state s' is not of interest, so I moved 'S' symbol into the set of states and ignored the head movement when s' = S. Taken that into account, there are 24 possible entries in each of the 10 places of the Turing program, given 24^{10} = 63,403,380,965,376 different Turing programs.

s	r	s'	w	m
1	ƀ	2	|	R
1	|	?	?	?
2	ƀ	?	?	?
2	|	?	?	?
3	ƀ	?	?	?
3	|	?	?	?
4	ƀ	?	?	?
4	|	?	?	?
5	ƀ	?	?	?
5	|	?	?	?

Fig. 6

Each TM must be started in state 1, and after reaching the 'stop state' the tape can be evaluated. If the TM does not stop, that has to be proved, i.e. with repeated execution interrupts and configuration analysis.

3.2 Making the Problem Smaller

Because solving the problem by examining all combinations with pencil and paper is illusive, I used a computer. Even with computer speed, the analysis of never-stopping TM's was too complicated, so for the sake of less runtime and easier programming, never-stopping TM's were neither analyzed nor proved, and the following decisions were made:

a) the tape is of fixed, finite size; if during the execution one of its borders is reached, it is assumed that the TM will expand infinitely in this direction,and so the TM is considered never-stopping.

b) the number of execution steps (until reaching the 'stop state') is limited. If a TM reaches this limit, it is assumed that the TM will run forever (even without needing infinite tape), and

so the TM is considered never-stopping. The computer runtime for simulating the TM's execution is proportional to the number of execution steps. When reaching the critical runtime limit('time-out'), the TM's execution is aborted.

c) the remaining Turing machines stop within the limits and are counted under various aspects. This situation is called 'Halt'.

In the following, the cause of ending the execution of a TM is a criterion to divide the TM's into these groups: End-of-tape, Time-out and Halt.

Processing all the 63.4 trillion TM's is too big a task when you have only about eight months of time; many reductions on the number of significant TM's are needed. When enumerating all TM's, sometimes not all 10 Turing program places need be filled, because they are never used during the TM's execution. My way of enumeration produces Turing programs with just the minimum number of Turing program places filled with some fixed entries; the others remain undefined. Obviously this avoidance of multiple processing similar TM's means a significant reduction of the number of TM's to be processed, perhaps a three orders of magnitude saving.

The top place (s=1, x=ƀ) can be constantly set to '2 | R' for the following reasons: This place is the first to be executed. If a ƀ were written onto the tape, the execution could as well begin with s=s' (the next state to be executed) instead of s=1, because the tape remains empty, only the head moves. There are two special cases: if s'=1, the TM will never stop; if m=S, the TM stops immediately. Starting the execution is s' instead of s is equal to exchanging the s and s' Turing program places, so 'ƀ' need not be written in the top place. At least, s'=2 is the only useful choice for s', because s'=1 builds a TM that never stops, and s'=3, 4,5 may be transformed to the s'=2-case by exchange of Turing program places.

3.3 The Main Algorithm

The computer proceeds as follows:

1) The TM execution starts; the remaining nine places are not yet fixed.

2) If the execution reaches a fixed place; that will be executed properly.

3) If the execution reaches a non-Fixed place, then all possible entries are made successively; for each the processing will continue recursiverly with step 2). Afterwards the place is made non-fixed again, and the execution situation is changed back to how it was at the previous situation 3). If there is no such situation, all TM combinations are processed and the algorithm ends.

4) If End-of-tape, Time-out, or Halt occured, the counters will be incremented appropriately and the execution situation is changed back to how it was at the previus situation 3).

Many processed TM's will reach End-of-tape, Time-out, or Halt with less than ten places fixed. In the tables presenting the results this will be a criterion for dividing all TM's into several groups.

All relevant TM's will be processed. There are further optimizations: The tenth place to be fixed shall - according to the problem - always write a '|' symbol, and stop (m=S). Therefore always '1 | S' will be used. To avoid place exchange symmetries, the "all possible entries" from algorithm step 3) is restricted to "all possible entries, where s' designates a state for which already one or two places are fixed, or - if existant - the least state for which no places are fixed".

3.4 Turing Machine Hardware

All programming was done on a personal computer (Apple II, 6502 microprocessor at 1 MHz, 47 K Bytes RAM), extended with a 8609 microprocessor board(running at the same speed) for programming convenience; the program, written in 6809 machine language, was about 700 Byte long. Comparisons with some bigger computers (PDP10, VAX 11) exhibited a runtime saving of less than 60% (on the VAX11), so I programmed the 6809 microprocessor. When calculations indicated that the program would run at least 20 months, I started building a Turing machine in hardware. Shown in appendix C and D, the Turing machine model is easy recognized in some TTL chips. With only 17 common available integrated circuits, a Turing program execution machine was build on a board that plugged into the personal computer.

After transferring a Turing program into the TURING PROGRAM STORAGE and starting the execution (under computer control), the extra hardware executes the program with about 4,500,000 steps per second (instead of 15,000 when using the microprocessor). The TM Control-Register provides information on whether the machine runs, or has stopped because of End-of-tape or Halt or Place-not-fixed. In the latter case, the information also contains the number of the place which had no fixed entry. During the hardware TM execution, the computer waits until the Time-out limit is reached. TM's that stopped were analyzed by the software on how many bars are written in how many steps; essentially this means a TM re-execution in software.

3.5 Performance

The tape size was 4096 cells, with the head on start position 2048. Time-out was raised after about 500,000 execution steps. These two limits are very high (a safety-factor of > 4), but the correct value of the Busy-Beaver-Function for n=5 may require even higher limits to be found. Nevertheless, the chances that the actual found TM is the solution is quite good (and the championship date

allowed no further extended computations). The TM execution with the hardware took 64% of the total 803 hours runtime. Increased speed could only have been achieved with special, expensive electronic parts.

3.6 Results

This is the best TM (in the sense of the problem):

s	r	s'	w	m
1	ƀ	2	|	R
1	|	3	ƀ	L
2	ƀ	3	|	R
2	|	4	|	R
3	ƀ	1	|	L
3	|	2	ƀ	R
4	ƀ	5	ƀ	R
4	|	1	|	S
5	ƀ	3	|	L
5	|	1	|	R

501 bars in 134,467 steps
found on 23.August 1982

Fig. 7

Using the hardware for investigating the Busy-Beaver-Function for six states, I found this TM:

s	r	s'	w	m
1	ƀ	2	|	R
1	|	1	|	S
2	ƀ	3	|	R
2	|	3	|	R
3	ƀ	4	|	R
3	|	6	|	L
4	ƀ	5	|	R
4	|	1	|	R
5	ƀ	6	|	R
5	|	4	ƀ	R
6	ƀ	2	|	L
6	|	3	ƀ	L

2075 bars in 4,208,824 steps
found on 23.December 1982

Fig. 8

If the 5-state TM from above is combined with a 7-state 'power stage', this Turing machine results:

s	r	s'	w	m
1	ƀ	2	|	R
1	|	3	ƀ	L
2	ƀ	3	|	R
2	|	4	|	R
3	ƀ	1	|	L
3	|	2	ƀ	R
4	ƀ	5	ƀ	R
4	|	6	|	L
5	ƀ	3	|	L
5	|	1	|	R
6	ƀ	7	|	L
6	|	6	|	R
7	ƀ	8	|	L
7	|	9	ƀ	R
8	ƀ	1	|	S
8	|	6	|	R
9	ƀ	10	|	L
9	|	9	|	R
10	ƀ	7	|	L
10	|	11	ƀ	R
11	ƀ	12	|	L
11	|	11	|	R
12	ƀ	10	|	L
12	|	12	|	L

Fig. 9

This TM produces extremely many bars:

$$6*(4096^{(4096^{(4096^{\cdots^{4}})})}),$$

where 4096 appears alltogether 166 times in the formula!

found on 29.November 1982

3.7 Statistics

TM's processed total: 126,891,605

Tape size: 4096 Cells, Time-out - abort after about 500000 steps.

Cause of end of TM execution

fixed	EndOfTape	Time-out	Halt	Total	Percentage	Runtime
10	68661869	13864419	28013532	110539820	87.11 %	802h55min
9	7640022	1506027	5526991	14673040	11.56 %	107h26min
8	664301	124627	733652	1522580	1.20 %	11h20min
7	55822	10365	76129	142316	0.11 %	1h05min
6	4489	751	7324	12564	0.01 %	6min
5	407	70	683	1160	0.00 %	1min
4	36	5	71	112	0.00 %	-
3	4	0	8	12	0.00 %	-
2	0	0	1	1	0.00 %	-
Total	77026950	15506264	34358391	126891605		
Percent.	60.70 %	12.22 %	27.08 %			

Fig. 10

4. Chasing the Busy Beaver, Part II

4.1 Terminology and Abbreviations

Cells which have been inspected by the TM are called used. Only a finite range of the tape can be used within a finite number of steps. This range is called the covered taped. A TM under test is called a candidate; if it does not stop, it is called a runner, if it does, the number of bars produced is called its output. If its output is not better than the output of other TM's tested before, it is called a looser; otherwise, it becomes the champion.

The Busy Beaver may be identical to the ultimate champion. At least, the champion's output is a lower bound for the Busy Beaver's output.

Two TM's are equivalent iff (if and only if) their tables can be transformed into each other by consistent renumbering of the states (and reordering of the lines). State 1, the starting state, must not be renumbered. This definition differs from that used by Wankmüller (look at Fig. 2), because he also considers those TM's to be equivalent which differ only in the starting state.

4.2 The Basic Algorithm, and the Need for Limits

There are 10 lines in the machine table of our candidates. For each line, there are
- 5 different states to follow,
- 2 different characters to be written, and
- 3 different moves,

making altogether 5 by 2 by 3 = 30 alternatives. Hence, there are 30 power 10 different tables.

While the general busy beaver function n=f(s), where s is the number of states and n the number of bars which a machine of s states can possibly produce and still stop, is noncomputable, for any particular s, the value might be found and proven to be correct. In practice, that was done for s up to 4. It is hard

to believe that the same kind of prove will ever be given for any s larger than 4. Thus, all attempts to find the 5-state Busy-Beaver must in some way test a large number of TM's.

Most of the candidates will not stop, so at some point in the computation a decision must be taken to regard a TM as a runner, which can be discarded. Obviously, this decision cannot be "clean", because the halting problem is not decidable. However, there is no other way to cope with the runners.

In practice, there are two values that can be used as simple criteria: the number of steps of the computation, and the range of the tape which was actually used in the computation.
In the approach described below, a step limit SL and a tape limit TL have been used. Any TM which did not stop within its first SL steps, or left the range of TL characters to the left or to the right of its starting position, was considered a runner. SL and TL are not necessarily fixed, but may depend on properties of the TM under test. A more sophisticated criterion is described in chapter 4.5.

Setting the limits is a most difficult task. On one hand, they determine the average execution time t (see chapter 4.6), on the other hand they must be wide enough to encompass the computation of the (unknown) Busy Beaver. Since this computation is presumably one of the very longest, nobody can ever be sure that they really do. To illustrate this difficulty, consider: The author's champion seemed to be safely within the limits (see figure 11), whilst the computation of the winner's champion exceeded both our SL and TL.

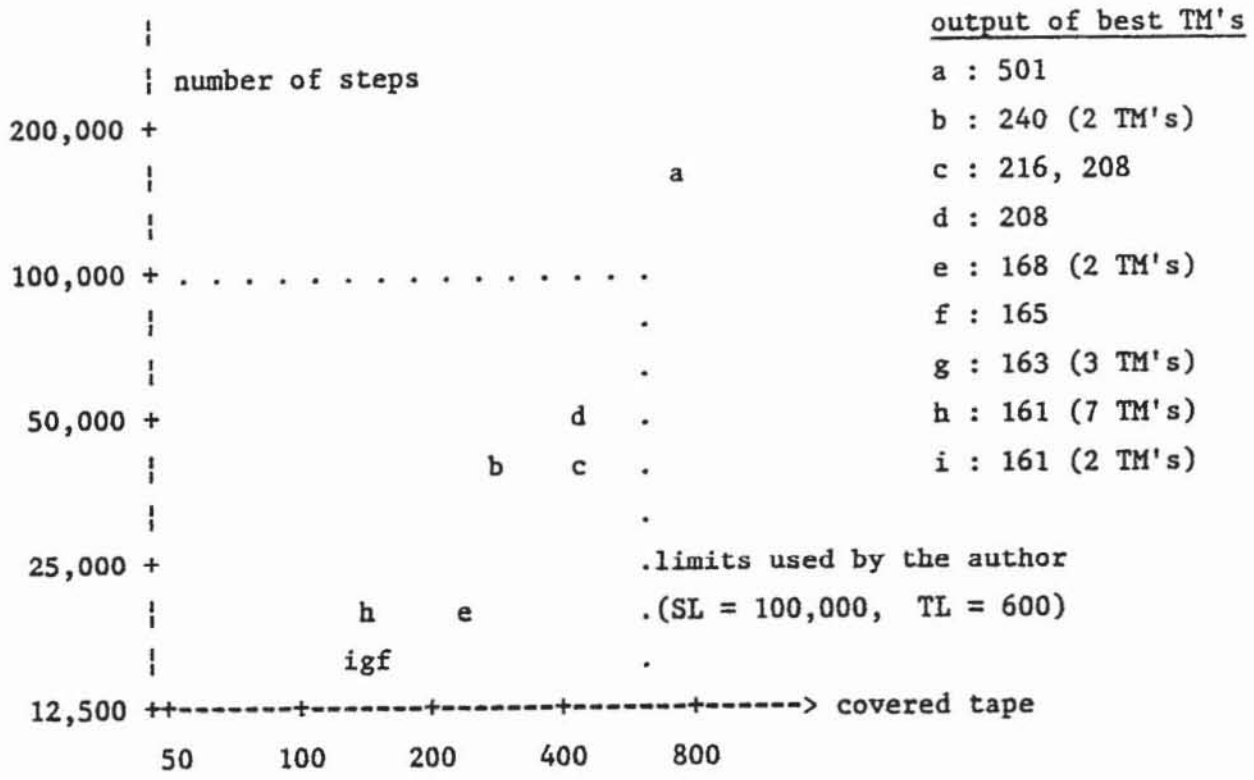


Fig. 11: Number of steps and covered tape for the 21 best TM's

4.3 Getting Rid of Redundant Beavers

The number 30 power 10 is very large: If every TM could be checked in 1 ms, the total check would last about 19,000 years. But the figures can be greatly reduced, because similar TM's form large equivalence classes. Therefore, the most important point is to find a complete, but non-redundand set of TM's. Some trivial statements form the basis for defining this set.

(a) Computations of equivalent TM's are identical except for the state-numbers. In particular, the number of steps, the covered tape, and the output are the same for equivalent TM's

(b) For every TM, there is onother (the symmetric TM) with the same, but mirrored computation. The only exception is the trivial case when the first step is a stop. Since there are very simple n-state TM's which can produce n bars, the trivial case may be excluded.

(c) In the first step of the computation, the state must be changed. Otherwise the TM will never stop.

(d) For every TM which performs n steps (n > 0) before writing a bar for the first time, there is another TM with the very same computation, except that the first n steps are missing. (This TM can be constructed from the first one by renaming its states so that the state in which the first bar is written becomes state "1").

(e) Let us call a line of a TM-table used iff it has been applied once or more during the computation. Then we can say that the content of unused lines is irrelevant.

(f) At most one line contaïning a stop ("stop-line") can be used. Thus, all other stop-lines are irrelevant.

(g) For every TM, which stops, but writes a blank in its last step, there exists a better one which differs only in that it writes a bar when it stops. The final state is irrelevant.

Statements a through h can be combined to form less trivial rules:

(u) The first line may be chosen to be 1 b: 2 | R. According to a, b, c and d, this covers all relevant cases.

(v) If a TM M1 is classified as a runner or a looser, then all other TM's which differ from M1 only in their unused lines are runners or loosers as well (see e).

(w) There is only one stop-line whose right hand side is 1 | S (see f, g).

From rules u through w, the algorithm for generating the non-redundant set of TM's can be easily deduced:

(1) Create a Turing table, of which all lines but the first one are empty.

s	r	s'	w	m
1	ƀ	2	|	R
1	|			
2	ƀ			
2	|			
3	ƀ			
3	|			
4	ƀ			
4	|			
5	ƀ			
5	|			

Fig. 12

(2) Start the computation in state 1 on the empty tape.

(3) Continue the computation as long as only defined lines of the table are used. If you run into a line whose right hand side is not yet defined, proceed as follows:

(3.1) If this is the last empty line, complete it as a stop line, and determine the number of bars on the tape. If the number is higher than the current champion, the new TM replaces the champion.

(3.2) If the actual line is not the last line, generate also the complete, but non-redundant set of right hand sides for the actual line, and continue the computation with each of them separately. The set contains all combinations of characters to be written(ƀ or |), moves (L or R), and distinguished states.

States which are unused (i.e. states of which both lines are unused) cannot be distinguished from each other, that means it is irrelevant which of them is chosen, because consistent renaming would permute them. Notice that the state for which the new line is generated is never unused.

Thus, the principle of generating the right hand sides becomes very simple: If the states n through 5 are unused, when a new right hand side is generated, the choice of states is 1 through n; if all states are used, all states have to be tried.

As an example, consider the first few steps of the algorithm:

After execution of line 1 ƀ, which is 1 ƀ: 2 | R, line 2 ƀ is to be used. Since it was not used before, its right hand side is undefined. There are 13 alternative solutions:

(0) 2 ƀ: 1 | S

(1) 2 ƀ: 1 | L (2) 2 ƀ: 1 | R (3) 2 ƀ: 1 | L (4) 2 ƀ: 1 ƀ R

(5) 2 ƀ: 2 | L (6) 2 ƀ: 2 | R (7) 2 ƀ: 2 | L (8) 2 ƀ: 2 ƀ R

(9) 2 ƀ: 3 | L (10) 2 ƀ: 3 | R (11) 2 ƀ: 3 | L (12) 2 ƀ: 3 ƀ R

Branch (0) produces 2 bars and becomes the current champion. When (2) is tried, the next line to be executed is again 1 ƀ, then 2 ƀ, ans so on, until step limit SL or tape limit TL is violated, and this case is discarded. Similar, (4), (6), and (8) will never reach any other unused line.
When (1) is tried, line 1 | is required, which is undefined. For that line, the same choice as above for 2 ƀ is checked (states 3, 4 and 5 are still unused), and again many of the alternatives fail immediately.

The algorithm given above can be understood as a searching meachnism in a tree structure. Wherever an unused line is met, the subtree is investigated, either until the TM stops, or until the subtree is descarded, because either of the limits SL and TL is violated. Discarding subtrees is really the big fish, because it means that less than 1% of all Turing tables which are neither equivalent nor symmetric, are actually tested.

The same basic algorithm as described above was applied by Uwe Schult, the winner of the Busy Beaver Competition.

By common sense rather than by any proof, we decided to insert a stop line only when all other lines are already used, because every line contributes to the complexity of the TM, and the Busy Beaver has to be a very clever TM. Therefore, in the example given above, case (0) was not checked.

The total effect of all teh improvements achieved by the generation algorithm was that only about 118 million TM's had to be checked. Compared with the figure 30**10, this means an improvement by a factor of ca. 4,600,00, which may be split into the effect of rule u (30), v (about 1800), and w (84). The effect of v can be further broken down into suppression of equivalent TM's (6) and dynamic generation of the Tuning table (ca. 300).

Another mechanism , though correct in principle, did not really contribute very much, because its effect was achieved by the dynamic geration anyway. In our algorithm, e.g., there was a test for certain cycles. If such a cycle occurs, the TM is a runner. See the following example:

Imagine that line 3 | is already defined, while line 3 ƀ is to be generated. Let the former be 3 |: 3 c1 m1 with arbitrary values for c1. and m1. Then, we must not generate 3 ƀ: 3 c2 m2, because the TM could never leave state 3. Our algorithm identified such (and more complex) runners without actually continuing the computation.

Similarly, an attempt was made to improve the efficiency by making SL depend on the number of unused lines. E.g., when only two lines are used, it is certainly not necessary to perform 100,000 steps before the TM is classified as a runner. Therefore, a list was used which provided different values for SL for every level of recursion. These values grow (approximately exponentially) with the number of used lines. Again, this measure did not contribute much because most of the time was spent in the very lowest level anyway.

4.4 Improving the Efficiency

Most programs follow the 80/20-rule ("20% of the code do 80% of the work"). In this case, the relation was rather 95 to 5, because performing the computations was by far the biggest task. This means that all the generation, checking etc. does not really matter. One step of the computation would cost us approximately 40 microseconds, which, after we had increased our limits SL and TL a couple of times, was too slow. Therefore, a fast, efficient computation algorithm (or the use of special hardware, as in Uwe Schult's solution) had to be developed. This was achieved by implementing the kernel in VAX-assembler, which reduced the execution time by a factor of 4 (see chapter 4.7).

One idea which was checked for its effect, but was not really introduced into our program, was to map every candidate onto an equivalent TM with a larger alphabet. By grouping, say, four cells into one "macro-cell", we have a TM with 16 characters. Since there are still five states, the macro table contains 5*16 lines. In fact, twice as many are required, because we have to distinguish if the macro-cell is entered from the left or from the right. Though working on the larger alphabet is alphabet is slower than just on | and ƀ, this effect is far outweighted by the fact that all the sequences of steps within one macro-cell become one "macro-step". Obviously, this approach does not pay if the macro-Turing-table is initialized in advance for every candidate, because many machines will stop, or fail, after a few steps anyway. But it is possible to build up the macro table "on the fly": Whenever the macro-TM enters a new cell, its content is checked. If it is not yet defined, the computation is performed as a sequence of normal steps, until the machine stops, or the cell is left. In the latter case, the result of the computation is loaded into the macro-table. For any particular candidate, only a couple of the 160 lines (for groups of four) are ever required.

In the PASCAL-implementation, working on groups of four cells resulted in an improvement by a factor of ca. 2.5. But this was with fairly low limits (SL = 50,000). Transferring this idea to our assembler program would have resulted in a considerable improvment.

4.5 Disqualifiing Lazy Beavers

When SL is large (say 50,000 or more), most of the machine time was wasted for those runners which do all the SL steps (i.e. which do not violate the tape limit before). By visual inspection of some of the computations by means of a program which displays the central part of the tape step by step on the terminal, it was found that many of these unproductive TM's exhibit a typical behaviour: The covered tape remains very narrow even after thousand of steps, because the TM loops within a couple of cells. In contrast, champions would grow in a christmas tree manner, i.e. they extend the covered tape alternating on both sides, usually on one side much faster than on the other.

To get rid of these "lazy beavers", an additional check was introduced: After a certain number of steps CL, the covered tape is computed, and the TM is classified as a runner, if the result is less than a lower bound LB. CL was chosen so that most of the loosers and those runners violating TL would fail without reaching this check. For TL = 600, this happens usually within thousand steps, which leeds to CL = 1000. The LB was set to 20, and since we did not feel quite sure about this empirical simplification, all the TM's just above LB (below 30) which still stop were stored and checked for their output. Their number turned out to be very small (about 70), and 81 was by far the best output in this group.

4.6 Influence of the Limits

If either SL or TL is small compared to the other, the smaller determines the average computation time t for one TM. A fairly good balance seems to be achieved if SL is of the same order as the square of TL, because of the Christmas tree structure of the computation described above. By eliminating the lazy beavers, the cost of SL was significantly reduced, so that TL was critical. Our final values (at the lowest level, see 3) were SL = 100,000 and TL = 600. With regard to Schult's results, both are just slightly too low, 150,000 and 700 would have done nicely, but not within the deadline of the competition.

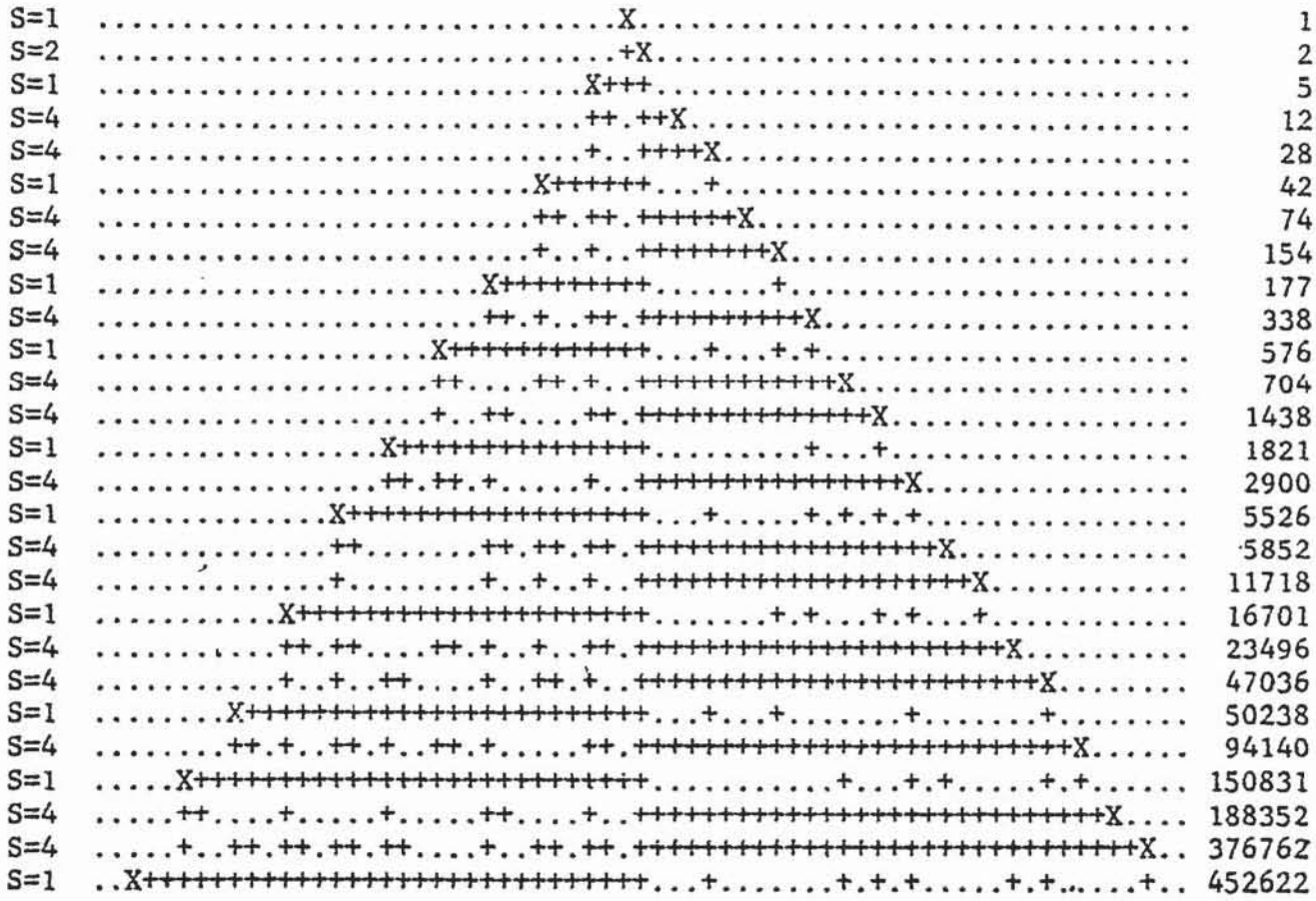


Fig. 13: Top of a (presumably) infinite Christmas tree

The influence of SL on t is linear for small TL, while the influence of TL is quadratic for large SL. For very large SL, the influence of TL becomes rather exponential, because a few very exotic TM's exhibit a very slow growth of the covered tape: If n cells are covered after s steps, n + 1 cells will be covered after 2n steps. Due to a bug in our program, such a pathologic TM once seemed to paralyse our algorithm. Its table is:

s	r	s'	w	m
1	ƀ	2	|	R
1	|	1	ƀ	L
2	ƀ	3	|	L
2	|	4	|	R
3	ƀ	1	|	S
3	|	4	|	L
4	ƀ	1	|	L
4	|	4	ƀ	R
5	ƀ	1	|	S
5	|	1	|	S

Fig. 14

Figure 13 shows the steps at which the covered tape has been extended. Instead of | and ƀ, the symbols "+" and "." are used. "X" indicates the position of the head, when it just writes a bar. At the left, the state is given, at the right the step-number.

4.7 Implementation

All the ideas described above were implemented in PASCAL on VAX/VMS. There is a procedure which performs the computation until an unused line is found. Then, the (local) generation prodedure is called. From there, the first procedure is called recursively. The use of recursive procedures makes it very easy to return to the root of the subtree when a branch is completed, because the whole status (including the tape) is passed to the next level as a value parameter.

In the assembler-version, the computation was actually started from the first step at each level, because that was altogether easier. Most of the time is spent in the lowest level anyway, where all but one of the lines of the Turing table have been generated. (ca. 97%).

At certain intervals (every 20,000 TM's, which means ca. every 1000 s), the program dumps status information in one of two files, in turn. When the program has to be restarted, it can automatically access those files.

Since the lines of the Turing table are not filled sequentially from top to bottom, it is fifficult to say whether or not a certain TM has already been checked, or in which order two TM's were found. For such purposes, a coding of the Turing table was defined, so that the code is steadily growing in lexical order while the tree is searched. Auxiliary programs allow the code to be translated into a normal table, and vice versa. See the appendix for examples.

The program would run at lowest priority on two VAX 11/780's, searching different parts of the tree. Thus, it just replaced the null-process for a couple of months.

4.8 Some Statistics

The complete tree of TM's has been searched several times, before the final limits were chosen. The last search took 1647 h (or 68,5 days) of CPU-time. According to our algorithm, four cases were distinguished:

(a) machines which stop,
(b) machines which violate the tape limit TL,
(c) lazy beavers (see chapter 4.5),
(d) machines which violate the step limit SL.

See Figure 15 below for some statistics.

case	Number	relative Number	average CPU-time/TM	relative CPU-time
(a)	28,013,513	23,7%	1.3 ms	0,6%
(b)	64,732,056	54,8%	34 ms	36,7%
(c)	22,334,845	18,9%	12,8 ms	4,8%
(d)	3,130,334	2,6%	1111 ms	57,9%
All	118,210,766	100,0%	50 ms	100,0%

Fig. 15: Statistics of a complete search

Of those TM's in group (a), about a third output just three or four bars. Above that, the number of TM's seems to decrease fairly steadily, but there are two significant maximums, one at ca. 65 bars, another one at ca. 161 bars. Uwe Schult, who did the same search but which much larger SL and TL, did not find any TM whose output lies between 501 bars (presumably the Busy Beaver) and 240 bars (our two champions).

4.9 Acknowledgements

First of all, I would like to express my gratitude to Prof. M.H. Rogers at Bristol University, England, where in 1969 I was not only allowed to collect my first (frustrating) experiences in programming, but also heard about the Busy Beaver. (That was, of course, the 4-state beaver.)

Secondly, I would like to thank many of my colleages who participated in this work, in particular Stefan Züger who implemented the assembler routine, Felix Kaufmann, who contributed some very helpful ideas, and Jiri Kriz who would often clarify the theoretical background.

Finally, I appreciate Frank Wankmüller's idea for this competition, from which we have learned a lot, and also had much fun.

5. Behaviour of Good Beavers

What is the temptation of the Busy Beaver Problem? The mathematicians have taught us that the general Busy Beaver function is non computable. Different from simple mathematical truths, this result pleases our brain, but not our heart (at least not mine). And though we know we cannot win the war against the mathematical law, we would like to win a battle, i.e. to find at least one particular Busy Beaver.

Therefore, we (just as the person who invented the name Busy Beaver) fail to look at the problem in a purely formal way. We try to apply feelings and experiences, because we want to understand the problem both in our brain an in our heart (which is, alas, not possible).

As part of this obstinate endeavour, I tried to learn how good Beavers work. I could not believe that such a simple system of ten lines in a Turing table can be applied in a way that one of the lines is only used after many thousand of steps. I tried to find a simple mechanism behind this fact.

Let us look at an example: Our champion (called TM-240) has this table:

s	r	s'	w	m
1	ƀ	2	|	R
1	|	1	|	R
2	ƀ	3	|	L
2	|	4	ƀ	R
3	ƀ	1	|	L
3	|	3	|	L
4	ƀ	1	|	S
4	|	5	|	R
5	ƀ	1	ƀ	R
5	|	2	|	R

Fig. 16

The start of its computation is shown in fig. 17. We can recognize the very regular behaviour. This pattern is emphasized in fig. 18, where only the lines are shown at which the covered tape has been extended.

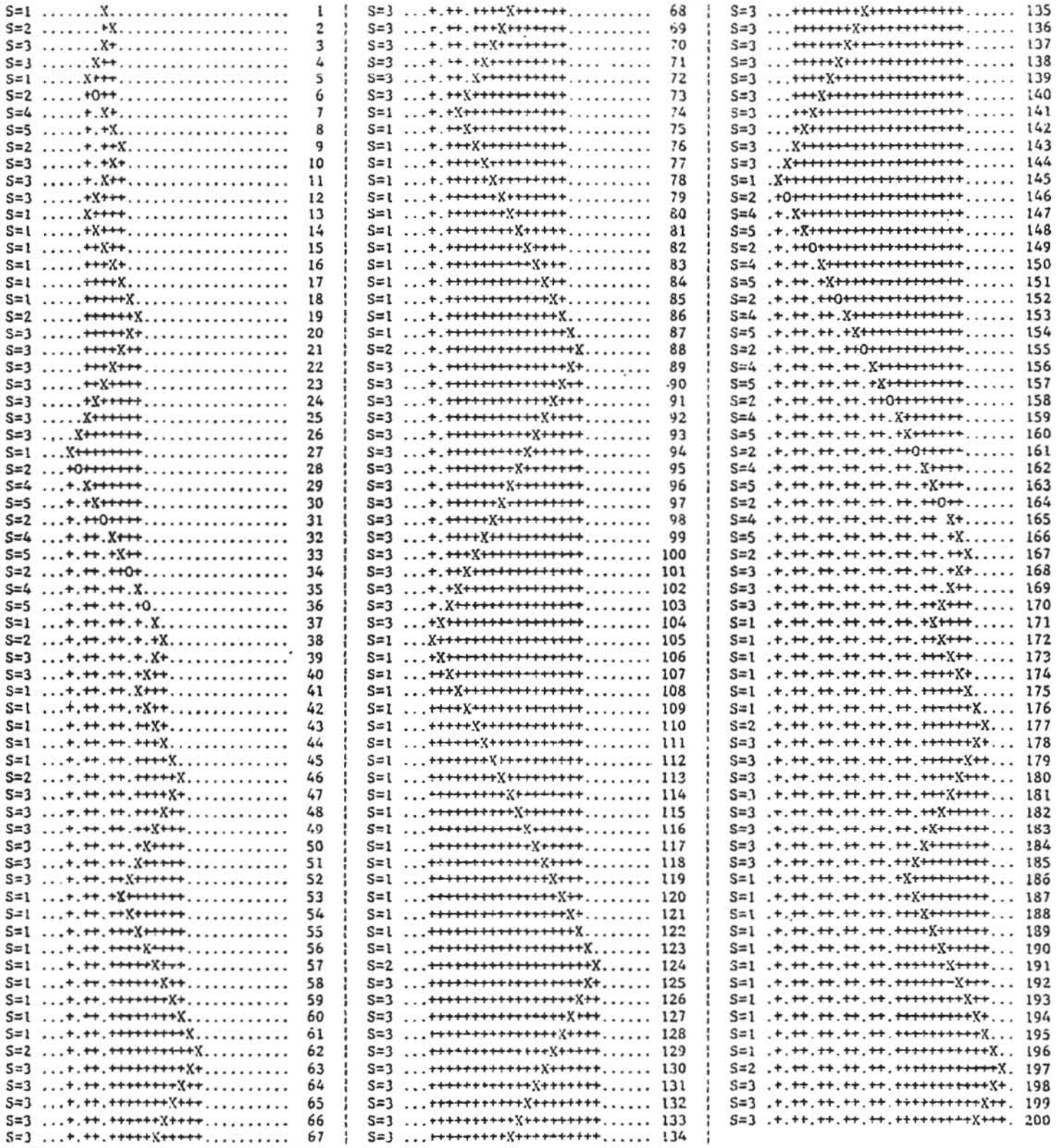


Fig. 17: First 200 (of 40,806) steps of TM-240.
("." is a blank, "+" is a bar. "X" and "O" indicate the head of the TM; "X" means a bar is being written, "O" means the TM is writing a blank. The state before execution of the step is given at the left.)

In fig. 18, we can easily see the structure of the computation: Groups of two bars, surrounding one blank ("+.+"), are replaced from right to left by a solid line of bars. For each group that is replaced, two more bars are written at the right end of the covered tape. When all the groups are consumed, the solid line is split again into groups of three cells, and the process starts again. There is only one situation when the computation may stop: When the line is split into groups, the end may be reached in three different states. Only two of these states permit the machine to continue, the third leads to the stop.

```
S=1 .........X.....................................................  1
S=2 .........+X....................................................  2
S=1 .......X+++....................................................  5
S=2 .......+.++X...................................................  9
S=1 .......X++++...................................................  13
S=2 .......++++++X.................................................  19
S=1 .....X++++++++.................................................  27
S=2 .....+.++.++.+.+X..............................................  38
S=2 .....+.++.++.+++++X............................................  46
S=2 .....+.++.++++++++++X..........................................  62
S=2 .....+.++++++++++++++X.........................................  88
S=2 .....++++++++++++++++++X.......................................  124
S=1 ...X++++++++++++++++++++.......................................  145
S=2 ...+.++.++.++.++.++.++.++X.....................................  167
S=2 ...+.++.++.++.++.++.+++++++X...................................  177
S=2 ...+.++.++.++.++.++++++++++++X.................................  197
S=2 ...+.++.++.++.++++++++++++++++X................................  227
S=2 ...+.++.++.++++++++++++++++++++++X.............................  267
S=2 ...+.++.++++++++++++++++++++++++++X............................  317
S=2 ...+.++++++++++++++++++++++++++++++++X.........................  377
S=2 ...++++++++++++++++++++++++++++++++++X.........................  447
S=1 .X++++++++++++++++++++++++++++++++++...........................  485
S=2 .+.++.++.++.++.++.++.++.++.++.++.++.++.+.+X....................  526
S=2 .+.++.++.++.++.++.++.++.++.++.++.++.++.+++++X..................  534
S=2 .+.++.++.++.++.++.++.++.++.++.++.++.++.++++++++++X..............  550
S=2 .+.++.++.++.++.++.++.++.++.++.++.++.++++++++++++++++X...........  576
S=2 .+.++.++.++.++.++.++.++.++.++.++.++++++++++++++++++++X..........  612
S=2 .+.++.++.++.++.++.++.++.++.++.++++++++++++++++++++++++++X.......  658
S=2 .+.++.++.++.++.++.++.++.++.++++++++++++++++++++++++++++++X......  714
S=2 .+.++.++.++.++.++.++.++.++++++++++++++++++++++++++++++++++X.....  780
S=2 .+.++.++.++.++.++.++++++++++++++++++++++++++++++++++++++++++X...  856
S=2 .+.++.++.++.++.++++++++++++++++++++++++++++++++++++++++++++++X.  942
S=2 .+.++.++.++.++++++++++++++++++++++++++++++++++++++++++++++++++X. 1038
S=2 .+.++.++.++++++++++++++++++++++++++++++++++++++++++++++++++++++X 1144
S=2 .+.++.++++++++++++++++++++++++++++++++++++++++++++++++++++++++++X 1260
S=2 .+.++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++X. 1386
```

Fig. 18: Compressed computation of TM-240.

The computation can be simulated by the following algorithm: Let M be a number, T a pair of two numbers g and n, and L, R, and B three functions, which are defined as follows:

```
B(T) = (2 * g) + 2          if n = 0, undefined otherwise
L(T) = (5*g) + (3*n) + 1    if n <> 0, undefined otherwise

R(M) = R(L(M DIV 3,2))      if M MOD 3 = 0
R(M) = R(L(M DIV 3,1))      if M MOD 3 = 1
R(M) = R  (M DIV 3,0)       if M MOD 3 = 2
```

The computation is defined by B(R(1)). It is shown in Figure 19.

M	L MOD 3	R.g	R.n	L	B
1	1	0	1	4	
4	1	1	1	9	
9	0	3	2	22	
22	1	7	1	39	
39	0	13	2	72	
72	0	24	2	127	
127	1	42	1	214	
214	1	71	1	359	
359	2	119	0		240

Fig. 19: Simulation of TM which outputs 240 bars.

Up to this point, the chapter seems to provide number-mystics rather than any logical argument. But the functions defined above have a very simple perspicuity: Let us concentrate on two special situtations, when either the covered tape contains nothing but bars (state 1), or when this solid line has been split up into groups, and the head of the TM has just reached the first blank at the right. Fig. 20 a shows the first of these lines.

For the lines marked by S=1, L gives the number of bars; for the other lines, g is the number of groups ("+.+"), while n indicates the situation at the end: n=1 means that the first blank is reached in state 2, when the second character of the group is to be written.

Correspondingly, n=2 means state 5 and no character in this group so far; n=0 means state 4 and two characters written. The latter case is the very end of the computation (see fig. 20b)

		L	R.g	R.n
S=1 .....X.....	1	1		
S=2 .....+X.....	2		0	1
S=1 .....X+++.....	5	4		
S=2 .....+.++X.....	9		1	1
S=1 .....X++++++++.....	27	9		
S=5 .....+.++.++.+0.....	36		3	2
S=1 ...X++++++++++++++++++.....	145	22		
S=2 ...+.++.++.++.++.++.++.++X.....	167		7	1
S=1 .X++++++++++++++++++++++++++++++++++....	485	39		
S=5 .+.++.++.++.++.++.++.++.++.++.++.++.++.+0...	524		13	2

Fig. 20 a: More compressed computation of TM-240.

S=4 ---++.++.++.++.++.++.++.++.++.X+++++.....	40800
S=5 ---++.++.++.++.++.++.++.++.++.+X++++.....	40801
S=2 ---++.++.++.++.++.++.++.++.++.++0+++.....	40802
S=4 ---++.++.++.++.++.++.++.++.++.++.X++.....	40803
S=5 ---++.++.++.++.++.++.++.++.++.++.+X+.....	40804
S=2 ---++.++.++.++.++.++.++.++.++.++.++0.....	40805
S=4 ---++.++.++.++.++.++.++.++.++.++.++.X.....	40806

	119	0

Fig. 20 b: The last steps of TM-240 (only part of the covered tape is shown)

The inner loops of the computation, which are suppressed by the algorithm given above, can be described in a very similar way. In fact, the algorithm was found using a bottom up technique, i.e. by starting from short cycles, and proceeding to more complex structures.

Other TM's can be described in a similar way. The TM which outputs 168 bars, e.g., is described by B(3) = 168:

```
B(M) = B(5 * (M DIV 3))        if M MOD 3 = 0
B(M) =  (2 * (M DIV 3)         if M MOD 3 = 1
B(M) = B(5 * (M DIV 3) + 9)    if M MOD 3 = 2
```

Schult's Busy Beaver was not thoroughly analysed. May be it is signigicant that its computation uses (different from all other good TM's, as far as they were analysed) MOD 4 rather than MOD 3.

With Z(M) = 3 * (M DIV 2), the output can be computed in eleven steps, starting from B(0):

```
B(M) = B(Z(M) + 6)          if M MOD 4 = 0
B(M) =  (Z(M) +6)/2         if M MOD 4 = 1
B(M) = B(Z(M) + 2)          if M MOD 4 = 2
B(M) = B(Z(M) + 13)         if M MOD 4 = 3
```

With these results, we can return to the question asked in the beginning of this chapter: The "trick" of good TM's is to build up complex computational steps from simpler ones. To me, it is much easier to believe that the complex algorithm can perform eleven steps before M MOD 4 happens to be 1 than that the original machine can do 134,467 steps before a certain line out of ten lines in its table happens to be applied.

6. A Beaver Typology

During our investigation, we found a couple of strange TM's. A few of them are shown below. According to their behaviour, we gave zoological names to them.

In the first group, there are various forms of diligent beavers. Beside the well known Busy Beaver, there is the careerist, who takes every effort to move forward as far as poosible. A third species is the beaver who works longer than any other Beaver (and still stops).
In practice, all three seem to be identical in the case of the 5-state TM.

In the second family are the non-productive beavers. Three species can be distinguished:

- The Civil Servant Beaver who cares most for his progress, but does not produce anything;
- the Scientific Beaver who does not produce anything either, but with more effort, and less effect on his position,
- and finally the Beaver Freak, who tries to survive as long as possible without producing anything, moving on the tape, and changing his state.

More formally, the definitions are as follow:

B : Number of bars output by the beaver
D : Distance of the cell where the beaver stops from the start
S : Number of steps until stop

CASTOR DILIGENTISSIMUS	(vulgo Busy Beaver)	: highest B
CASTOR PRIMUS	(vulgo Place-hunter Beaver)	: highest D
CASTOR PERPETUUS	(vulgo Swabian Beaver)	: highest S
CASTOR MINISTERIALIS	(vulgo Civil Servant Beaver)	: B=0 AND highest D
CASTOR SCIENTIFICUS	(vulgo Scientfic Beaver)	: B=0 AND highest S
CASTOR EXFLIPPUS	(vulgo Beaver-Freak)	: B=0 AND D=0 AND initial state = final state AND highest S

And here are the various champions; the first one is, of course, Uwe Schult's result.

Species: CASTOR ET DILIGENTISSIMUS ET PRIMUS ET PERPETUUS
(alias CASTOR SCHULTIS)

Stops in step 134467 at position 656 with 501 bars between -10 and 656

s	r	s'	w	m
1	ƀ	2	|	R
1	|	3	ƀ	L
2	ƀ	3	|	R
2	|	4	|	R
3	ƀ	1	|	L
3	|	2	ƀ	R
4	ƀ	5	ƀ	R
4	|	1	|	S
5	ƀ	3	|	L
5	|	1	|	R

Fig. 21a

Species: CASTOR MINISTERIALIS

Both TM's stop in step 52 at position 11, no bars on the tape.

s	r	s'	w	m
1	ƀ	2	|	R
1	|	1	|	R
2	ƀ	3	|	R
2	|	5	ƀ	R
3	ƀ	4	|	L
3	|	1	ƀ	R
4	ƀ	2	|	L
4	|	4	|	L
5	ƀ	1	ƀ	S
5	|	2	ƀ	R

s	r	s'	w	m
1	ƀ	2	ƀ	R
1	|	1	ƀ	L
2	ƀ	3	ƀ	R
2	|	4	ƀ	R
3	ƀ	1	|	L
3	|	5	|	R
4	ƀ	3	|	R
4	|	1	ƀ	S
5	ƀ	4	|	R
5	|	3	|	R

Fig. 21b

Species: CASTOR SCIENTIFICUS

Stops in step 187 at position 8, no bars on the tape.

s	r	s'	w	m
1	ƀ	2	ƀ	R
1	|	1	ƀ	L
2	ƀ	3	ƀ	R
2	|	1	ƀ	S
3	ƀ	4	|	R
3	|	5	|	L
4	ƀ	1	|	L
4	|	4	ƀ	L
5	ƀ	3	|	R
5	|	5	|	R

Fig. 21c

The Computation of CASTOR SCIENTIFICUS is shown in fig. 22

Species: CASTOR EXFLIPPUS

Stops in step 67 at position 0, no bars on the tape.

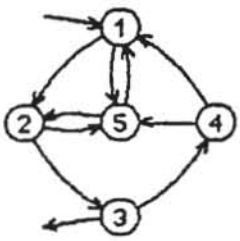
s	r	s'	w	m
1	ƀ	2	ƀ	R
1	|	1	ƀ	L
2	ƀ	3	|	R
2	|	1	ƀ	S
3	ƀ	3	ƀ	L
3	|	4	|	R
4	ƀ	4	ƀ	L
4	|	5	|	R
5	ƀ	1	|	L
5	|	5	ƀ	L

Fig. 21d

```
S=1 .0.....  1
S=2 ..0.....  2
S=3 ...X.....  3
S=4 ...+X.....  4
S=1 ...0+.....  5
S=1 ..0.+.....  6
S=2 ...0+.....  7
S=3 ....X.....  8
S=5 ...X+.....  9
S=3 ...+X..... 10
S=5 ...X+..... 11
S=5 ...+X..... 12
S=5 ...++X..... 13
S=3 ...+++X..... 14
S=4 ...++++X..... 15
S=1 ...+++0+..... 16
S=1 ...++0.+..... 17
S=1 ...+0..+..... 18
S=1 ...0...+..... 19
S=1 ..0....+..... 20
S=2 ...0...+..... 21
S=3 ....X..+..... 22
S=4 ....+X.+..... 23
S=1 ....0+.+..... 24
S=1 ...0.+.+..... 25
S=2 ....0+.+..... 26
S=3 .....X.+..... 27
S=5 ....X+.+..... 28
S=3 ....+X.+..... 29
S=5 ....X+.+..... 30
S=5 ....+X.+..... 31
S=5 ....++X+..... 32
S=3 ....+++X..... 33
S=5 ....++X+..... 34
S=5 ....+++X..... 35
S=5 ....++++X..... 36
S=3 ....+++++X..... 37
S=4 ....++++++X..... 38
S=1 ....+++++0+..... 39
S=1 ....++++0.+..... 40
S=1 ....+++0..+..... 41
S=1 ....++0...+..... 42
S=1 ....+0....+..... 43
S=1 ....0.....+..... 44
S=1 ...0......+..... 45
S=2 ....0.....+..... 46
S=3 .....X....+..... 47
S=4 .....+X...+..... 48
S=1 .....0+...+..... 49
S=1 ....0.+...+..... 50
S=2 .....0+...+..... 51
S=3 ......X...+..... 52
S=5 .....X+...+..... 53
S=3 .....+X...+..... 54
S=5 .....X+...+..... 55
S=5 .....+X...+..... 56
S=5 .....++X..+..... 57
S=3 .....+++X.+..... 58
S=4 .....++++X+..... 59
S=1 .....+++0++..... 60
S=1 .....++0.++..... 61
S=1 .....+0..++..... 62
S=1 .....0...++..... 63
S=1 ....0....++..... 64
S=2 .....0...++..... 65
S=3 ......X..++..... 66
S=4 ......+X.++..... 67
S=1 ......0+.++..... 68
S=1 .....0.+.++..... 69
S=2 ......0+.++..... 70
S=3 .......X.++..... 71
S=5 ......X+.++..... 72
S=3 ......+X.++..... 73
S=5 ......X+.++..... 74
S=5 ......+X.++..... 75
S=5 ......++X++..... 76
S=3 ......+++X+..... 77
S=5 ......++X++..... 78
S=5 ......+++X+..... 79
S=5 ......++++X..... 80
S=5 ......+++++X..... 81
S=3 ......++++++X..... 82
S=4 ......+++++++X..... 83
S=1 ......++++++0+..... 84
S=1 ......+++++0.+..... 85
S=1 ......++++0..+..... 86
S=1 ......+++0...+..... 87
S=1 ......++0....+..... 88
S=1 ......+0.....+..... 89
S=1 .....0......+..... 90
S=1 .....0.......+..... 91
S=2 ......0......+..... 92
S=3 .......X.....+..... 93
S=4 ......+X....+..... 94
S=1 .......0+....+..... 95
S=1 ......0.+....+..... 96
S=2 .......0+....+..... 97
S=3 ........X....+..... 98
S=5 .......X+....+..... 99
S=3 .......+X....+..... 100
S=5 .......X+....+..... 101
S=5 .......+X....+..... 102
S=5 .......++X...+..... 103
S=3 .......+++X..+..... 104
S=4 .......++++X.+..... 105
S=1 .......+++0+.+..... 106
S=1 .......++0.+.+..... 107
S=1 .......+0..+.+..... 108
S=1 .......0...+.+..... 109
S=1 ......0....+.+..... 110
S=2 .......0...+.+..... 111
S=3 ........X..+.+..... 112
S=4 ........+X.+.+..... 113
S=1 ........0+.+.+..... 114
S=1 .......0.+.+.+..... 115
S=2 ........0+.+.+..... 116
S=3 .........X.+.+..... 117
S=5 ........X+.+.+..... 118
S=3 ........+X.+.+..... 119
S=5 ........X+.+.+..... 120
S=5 ........+X.+.+..... 121
S=5 ........++X+.+..... 122
S=3 ........+++X.+..... 123
S=5 ........++X+.+..... 124
S=5 ........+++X.+..... 125
S=5 ........++++X+..... 126
S=3 ........+++++X..... 127
S=5 ........++++X+..... 128
S=5 ........+++++X..... 129
S=5 ........++++++X.... 130
S=3 ........+++++++X... 131
S=4 ........++++++++X.. 132
S=1 ........+++++++0+.. 133
S=1 ........++++++0.+.. 134
S=1 ........+++++0..+.. 135
S=1 ........++++0...+.. 136
S=1 ........+++0....+.. 137
S=1 ........++0.....+.. 138
S=1 ........+0......+.. 139
S=1 ........0.......+.. 140
S=1 .......0........+.. 141
S=2 ........0.......+.. 142
S=3 .........X......+.. 143
S=4 .........+X.....+.. 144
S=1 .........0+.....+.. 145
S=1 ........0.+.....+.. 146
S=2 .........0+.....+.. 147
S=3 ..........X.....+.. 148
S=5 .........X+.....+.. 149
S=3 .........+X.....+.. 150
S=5 .........X+.....+.. 151
S=5 .........+X.....+.. 152
S=5 .........++X....+.. 153
S=3 .........+++X...+.. 154
S=4 .........++++X..+.. 155
S=1 .........+++0+..+.. 156
S=1 .........++0.+..+.. 157
S=1 .........+0..+..+.. 158
S=1 .........0...+..+.. 159
S=1 ........0....+..+.. 160
S=2 .........0...+..+.. 161
S=3 ..........X..+..+.. 162
S=4 ..........+X.+..+.. 163
S=1 ..........0+.+..+.. 164
S=1 ..........0.+.+..+.. 165
S=2 ..........0+.+..+.. 166
S=3 ...........X.+..+.. 167
S=5 ...........X+.+..+.. 168
S=3 ...........+X.+..+.. 169
S=5 ...........X+.+..+.. 170
S=5 ...........+X.+..+.. 171
S=5 ...........++X+..+.. 172
S=3 ...........+++X..+.. 173
S=5 ...........++X+..+.. 174
S=5 ...........+++X..+.. 175
S=5 ...........++++X.+.. 176
S=3 ...........+++++X+.. 177
S=4 ...........++++++0.. 178
S=4 ...........+++++0... 179
S=4 ...........++++0.... 180
S=4 ...........+++0..... 181
S=4 ...........++0...... 182
S=4 ...........+0....... 183
S=4 ...........0........ 184
S=4 .........X......... 185
S=1 .........0+........ 186
S=2 .........0......... 187
```

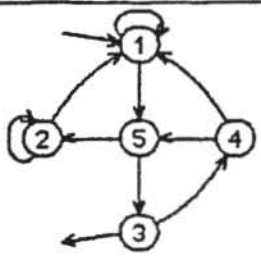
Fig. 22: Computation of the CASTOR SCIENTIFICUS

Appendix A: Examples of TMs, which produce many bars



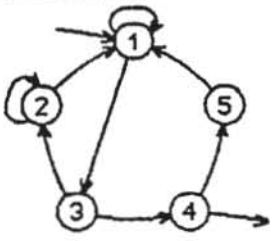
zr	wmz'
10	1R2
11	0L5
20	1R5
21	1R3
30	0R4
31	1LH
40	1L5
41	1R1
50	1L1
51	0R2

501 in 134467



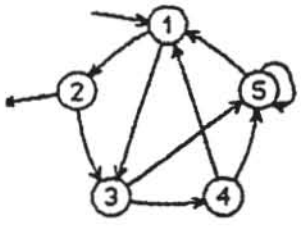
zr	wmz'
10	1R5
11	1R1
20	1L1
21	1L2
30	1LH
31	1R4
40	0R1
41	1R5
50	1L2
51	0R3

240 in 40806



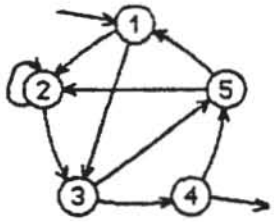
zr	wmz'
10	1R3
11	1R1
20	1L1
21	1L2
30	1L2
31	0R4
40	1LH
41	1R5
50	0R1
51	0L1

240 in 41360



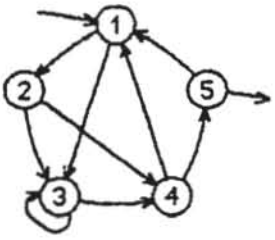
zr	wmz'
10	1R2
11	0L3
20	0R3
21	1LH
30	1R4
31	0R5
40	1L5
41	0R1
50	0L1
51	0L5

216 in 40899 *



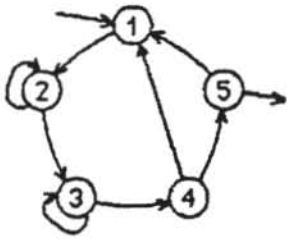
zr	wmz'
10	1R2
11	0L3
20	0R3
21	0R2
30	1L4
31	0R5
40	0L5
41	1LH
50	1L1
51	0L2

208 in 39734



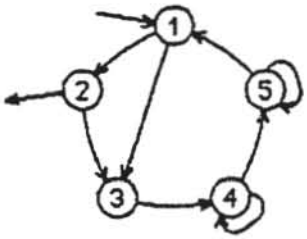
zr	wmz'
10	1R2
11	0R3
20	1L3
21	0R4
30	0L4
31	0L3
40	1R5
41	0L1
50	0R1
51	1LH

208 in 49249



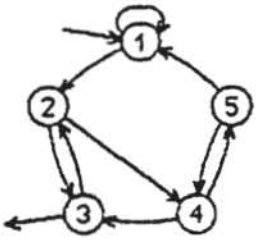
zr	wmz'
10	1R2
11	0L2
20	1L3
21	1L2
30	0R4
31	1R3
40	1L1
41	1R5
50	1LH
51	0R1

168 in 21286



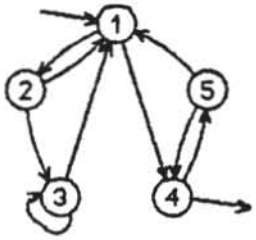
zr	wmz'
10	1R3
11	1L2
20	1LH
21	0L3
30	1L4
31	0R4
40	1R5
41	1R4
50	0L1
51	1L5

168 in 21294



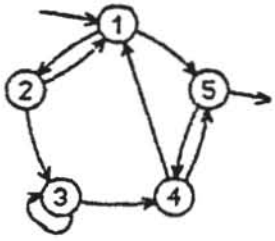
zr	wmz'
10	1R2
11	1L1
20	1R4
21	1L3
30	1LH
31	0L2
40	1R5
41	1R3
50	0L1
51	1R4

165 in 15589



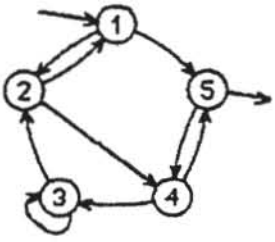
zr	wmz'
10	1R2
11	1R4
20	1L3
21	1R1
30	1R1
31	1L3
40	1LH
41	0L5
50	1R1
51	1L4

163 in 14965



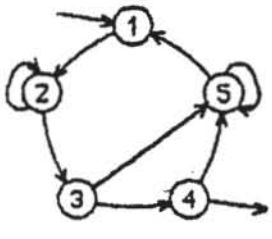
zr	wmz'
10	1R2
11	1R5
20	0L3
21	1R1
30	1L4
31	1L3
40	1R1
41	1L5
50	1LH
51	0L4

163 in 14975



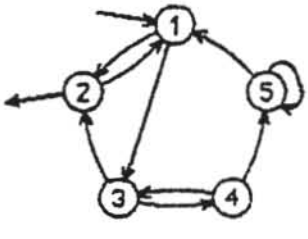
zr	wmz'
10	1R2
11	1R5
20	1R4
21	1R1
30	0R2
31	1L3
40	1L3
41	1L5
50	1LH
51	0L4

163 in 15271



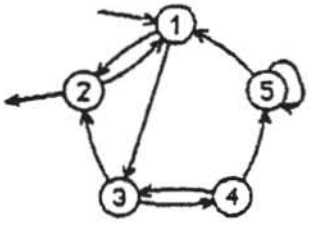
zr	wmz'
10	1R2
11	0L2
20	1R3
21	1L2
30	1L5
31	0R4
40	1LH
41	0R5
50	1L5
51	1L1

162 in 41710



zr	wmz'
10	1R3
11	1L2
20	1LH
21	0L1
30	1R4
31	1R2
40	1L5
41	1R3
50	0L1
51	1L5

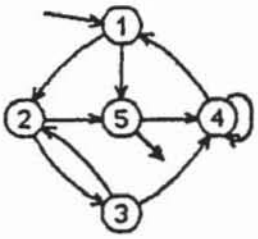
161 in 14371



zr	wmz'
10	1R3
11	1L2
20	1LH
21	0L1
30	1R4
31	1R2
40	1L5
41	1R3
50	1R1
51	1L5

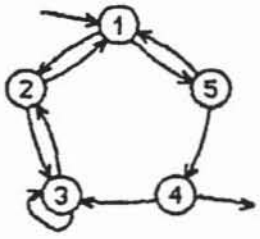
161 in 14389

*Remark: Choosing state 2,3,4,5 as initial state, this machine produces 208,208,208,216 bars, respectivly



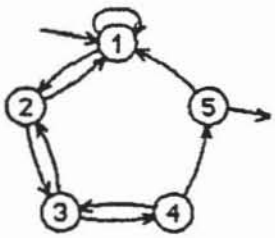
zr	wmz'
10	1R2
11	1L5
20	1R3
21	1R5
30	1L4
31	1R2
40	0L1
41	1L4
50	1LH
51	0R4

161 in 20398



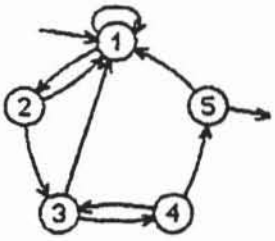
zr	wmz'
10	1R2
11	1L5
20	1L1
21	0R3
30	1R2
31	1L3
40	1LH
41	0L3
50	1L1
51	0L4

161 in 20711



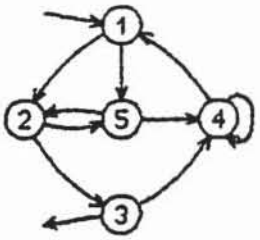
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R1
30	1R2
31	1L4
40	1L3
41	0L5
50	1LH
51	0L1

161 in 20711



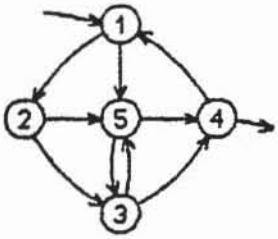
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R1
30	0R1
31	1L4
40	1L3
41	0L5
50	1LH
51	0L1

161 in 20729



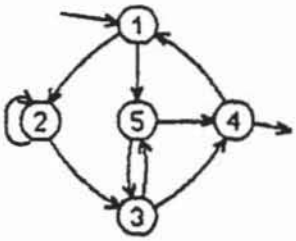
zr	wmz'
10	1R2
11	0R5
20	1R5
21	1R3
30	1LH
31	0R4
40	0L1
41	1R4
50	1L4
51	1R2

161 in 20796



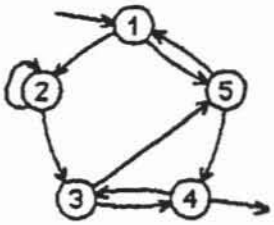
zr	wmz'
10	1R2
11	1L5
20	1L5
21	1R3
30	1L5
31	0L4
40	1LH
41	0R1
50	1R4
51	1L3

161 in 22059



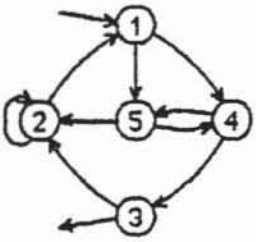
zr	wmz'
10	1R2
11	1L5
20	0L2
21	1R3
30	1L5
31	0L4
40	1LH
41	0R1
50	1R4
51	1L3

161 in 22335



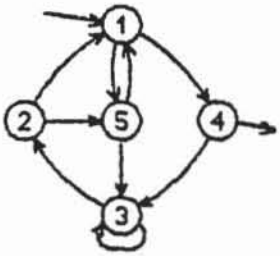
zr	wmz'
10	1R2
11	1L5
20	1R3
21	1R2
30	1L5
31	1R4
40	1LH
41	0R3
50	1L1
51	1L4

159 in 13821



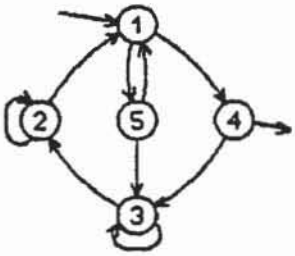
zr	wmz'
10	1R5
11	1L4
20	1L1
21	1L2
30	1LH
31	0L2
40	1R5
41	0R3
50	1L2
51	1R4

159 in 20492



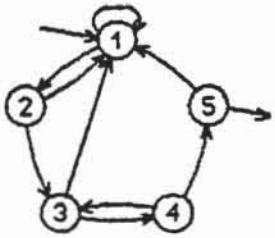
zr	wmz'
10	1R5
11	0R4
20	1R5
21	1L1
30	1L2
31	1L3
40	1LH
41	0L3
50	1L3
51	1R1

159 in 20492



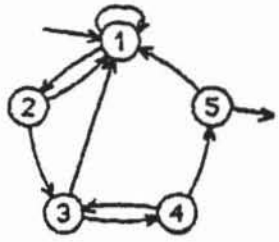
zr	wmz'
10	1R5
11	0R4
20	0R2
21	1L1
30	1L2
31	1L3
40	1LH
41	0L3
50	1L3
51	1R1

159 in 20766



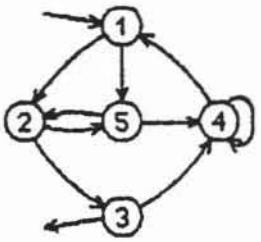
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R1
30	1L1
31	1L4
40	1L3
41	0L5
50	1LH
51	0L1

157 in 19129



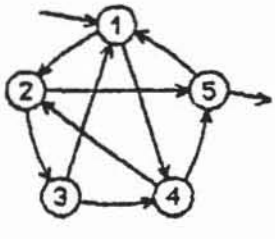
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R1
30	1R1
31	1L4
40	1L3
41	0L5
50	1LH
51	0L1

157 in 19145



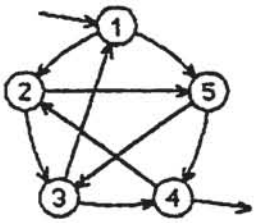
zr	wmz'
10	1R2
11	0L5
20	1L5
21	0L3
30	1LH
31	0L4
40	0R1
41	1L4
50	1R4
51	1L2

157 in 19463



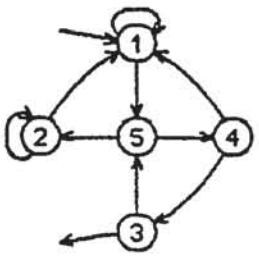
zr	wmz'
10	1R2
11	1L4
20	1R3
21	0L5
30	1L1
31	1R4
40	1R5
41	0L2
50	1LH
51	0R1

157 in 33174



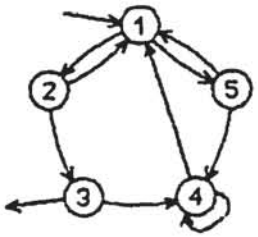
zr	wmz'
10	1R2
11	1L5
20	1L3
21	1R5
30	1L1
31	0R4
40	1LH
41	0L2
50	1L4
51	0R3

157 in 33178



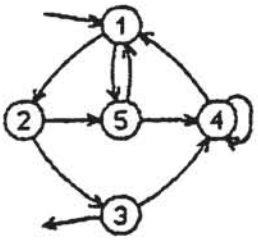
zr	wmz'
10	1R5
11	0L1
20	1L2
21	1L1
30	1LH
31	0R5
40	1R1
41	1R3
50	0R2
51	1R4

154 in 20739



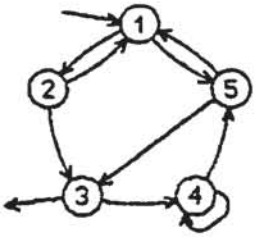
zr	wmz'
10	1R2
11	0L5
20	1L1
21	1R3
30	0R4
31	1LH
40	0R1
41	1R4
50	1L1
51	1L4

125 in 23838



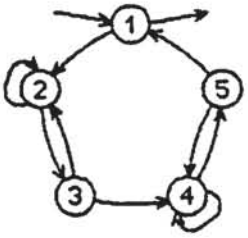
zr	wmz'
10	1R2
11	0L5
20	1L5
21	1R3
30	0R4
31	1LH
40	0R1
41	1R4
50	1L1
51	1L4

125 in 24778



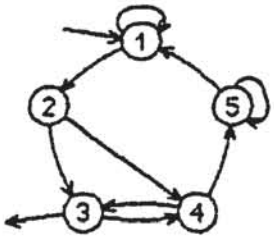
zr	wmz'
10	1R2
11	0L5
20	1R3
21	0R1
30	1L4
31	1LH
40	1L5
41	1L4
50	1L1
51	0L3

112 in 6147



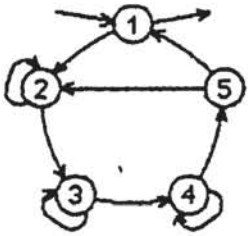
zr	wmz'
10	1R2
11	1LH
20	1L2
21	0R3
30	0R2
31	1L4
40	0L5
41	1L4
50	1L1
51	0L4

106 in 5012



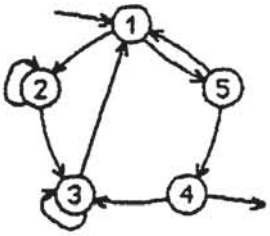
zr	wmz'
10	1R2
11	1R1
20	1R4
21	1L3
30	1LH
31	0R4
40	1L5
41	1R3
50	1L1
51	1L5

105 in 10707



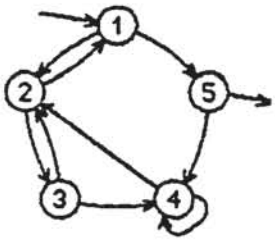
zr	wmz'
10	1R2
11	1LH
20	0R3
21	0R2
30	0R4
31	1R3
40	1L4
41	1L5
50	0L1
51	0R2

99 in 2411



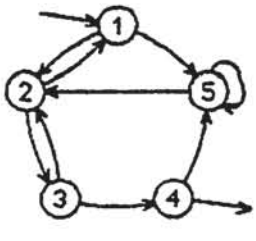
zr	wmz'
10	1R2
11	0L5
20	0L3
21	0R2
30	1R1
31	1L3
40	0L3
41	1LH
50	0L4
51	1L1

99 in 18040



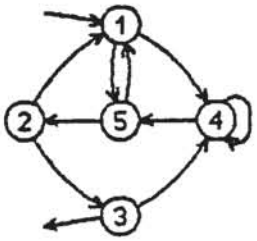
zr	wmz'
10	1R2
11	1L5
20	1L1
21	0R3
30	1R2
31	1R4
40	0L2
41	1L4
50	0L4
51	1LH

97 in 14095



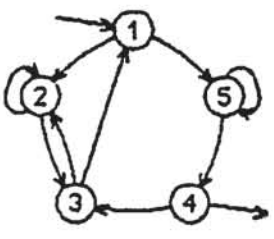
zr	wmz'
10	1R2
11	1R5
20	1L3
21	0R1
30	1R2
31	1L4
40	0L5
41	1LH
50	0L2
51	1L5

97 in 14095



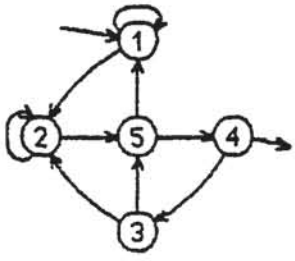
zr	wmz'
10	1R5
11	1R4
20	1R1
21	1L3
30	0L4
31	1LH
40	0L5
41	1L4
50	1L2
51	0R1

97 in 14661



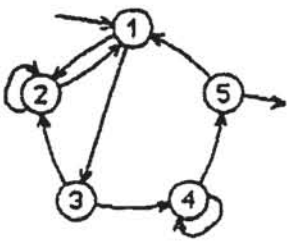
zr	wmz'
10	1R2
11	0R5
20	1L3
21	0L2
30	1R1
31	0L2
40	0R3
41	1LH
50	1R5
51	1R4

91 in 8245



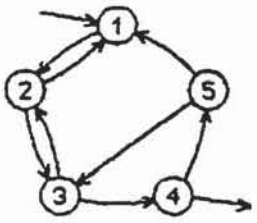
zr	wmz'
10	1R2
11	0R1
20	1L5
21	0L2
30	0L5
31	1L2
40	1L3
41	1LH
50	1R1
51	1L4

89 in 5685



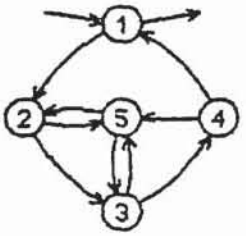
zr	wmz'
10	1R3
11	0L2
20	1L1
21	0L2
30	1R2
31	0R4
40	1R4
41	1R5
50	0R1
51	1LH

88 in 7706



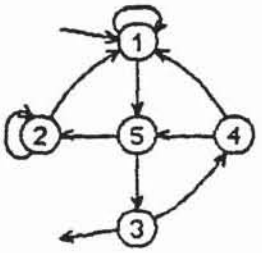
zr	wmz'
10	1R2
11	0R2
20	1L3
21	0R1
30	1R4
31	0L2
40	1LH
41	1R5
50	0R1
51	1R3

87 in 3189



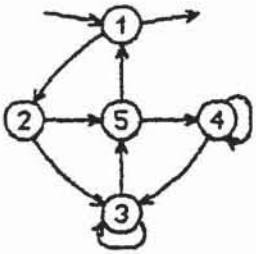
zr	wmz'
10	1R2
11	1LH
20	1R5
21	1R3
30	0R4
31	0L5
40	1L5
41	1R1
50	1L3
51	0R2

83 in 3719



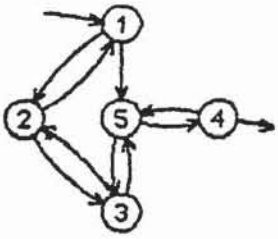
zr	wmz'
10	1R5
11	0R1
20	1L1
21	0L2
30	1R4
31	1LH
40	0R5
41	1R1
50	1L2
51	1R3

83 in 5067



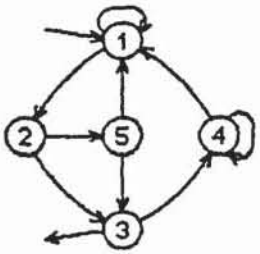
zr	wmz'
10	1R2
11	1LH
20	0R5
21	1R3
30	1R5
31	0R3
40	1L3
41	0L4
50	1L4
51	1R1

83 in 5074



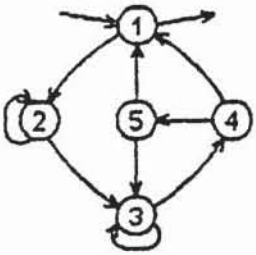
zr	wmz'
10	1R5
11	1L2
20	1R3
21	0L1
30	1L5
31	1R2
40	1LH
41	1L5
50	0R3
51	1L4

83 in 5309



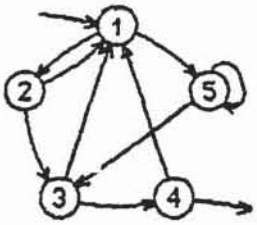
zr	wmz'
10	1R2
11	0L1
20	0R5
21	1R3
30	1L4
31	1LH
40	1R1
41	0L4
50	0L3
51	0R1

83 in 14264



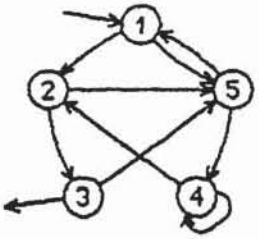
zr	wmz'
10	1R2
11	1LH
20	1L3
21	0R2
30	1L4
31	0R3
40	0L5
41	1L1
50	0R1
51	0L3

83 in 14268



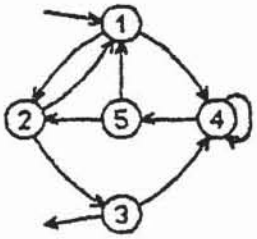
zr	wmz'
10	1R2
11	0R5
20	1R3
21	1R1
30	0L4
31	0L1
40	1LH
41	1L1
50	1L3
51	1R5

83 in 16328



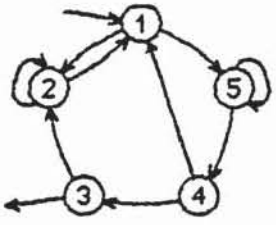
zr	wmz'
10	1R2
11	1R5
20	0L3
21	0L5
30	1LH
31	1L5
40	1L2
41	1R4
50	1R1
51	0R4

83 in 16351



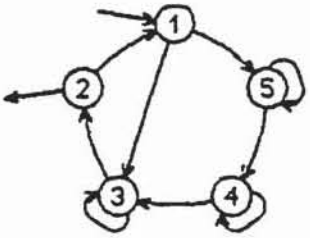
zr	wmz'
10	1R2
11	1L4
20	1R3
21	1L1
30	0R4
31	1LH
40	1R5
41	0L4
50	1L2
51	0R1

81 in 2636



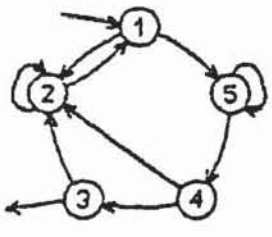
zr	wmz'
10	1R2
11	0L5
20	1L1
21	1R2
30	1LH
31	0R2
40	1L1
41	1R3
50	1R5
51	1R4

81 in 5230



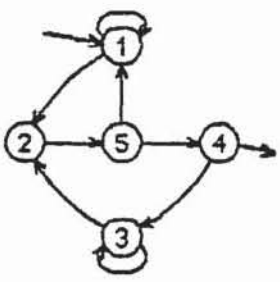
zr	wmz'
10	1R3
11	0L5
20	1LH
21	1L1
30	1L2
31	1R3
40	1R4
41	0R3
50	1L4
51	1L5

81 in 5323



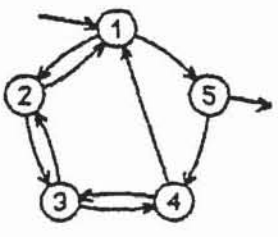
zr	wmz'
10	1R2
11	0L5
20	1L1
21	1R2
30	1LH
31	0R2
40	0L2
41	1R3
50	1R5
51	1R4

81 in 7682



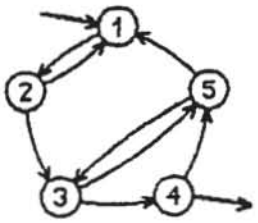
zr	wmz'
10	1R2
11	0R1
20	1L5
21	0R5
30	1L2
31	0R3
40	0L3
41	1LH
50	1R1
51	1L4

81 in 9865



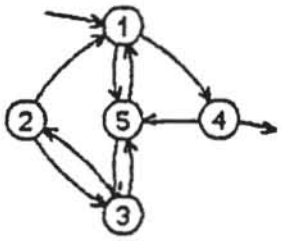
zr	wmz'
10	1R2
11	1R5
20	1L3
21	0R1
30	0L4
31	1L2
40	0R1
41	1L3
50	1LH
51	0L4

81 in 14370



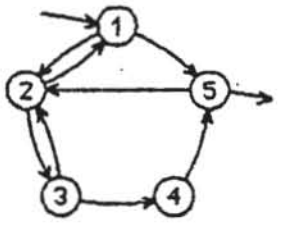
zr	wmz'
10	1R2
11	1L2
20	1L1
21	0R3
30	0L5
31	1R4
40	1LH
41	0L5
50	0R1
51	1L3

81 in 21010



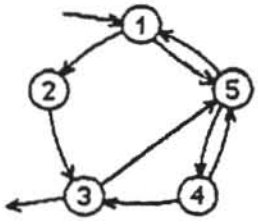
zr	wmz'
10	1R5
11	0R4
20	0R1
21	1L3
30	0L2
31	1L5
40	1LH
41	0L5
50	1L3
51	0R1

80 in 14370



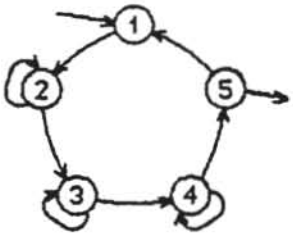
zr	wmz'
10	1R2
11	0R5
20	1L3
21	0R1
30	1L4
31	1L2
40	1R5
41	0R5
50	1LH
51	0L2

80 in 21040



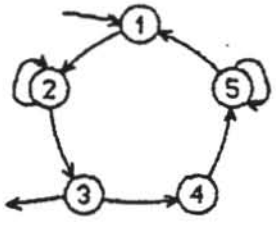
zr	wmz'
10	1R2
11	1R5
20	1L3
21	0L3
30	1LH
31	0R5
40	1L5
41	0L3
50	1R1
51	0L4

80 in 21044



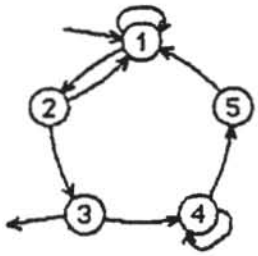
zr	wmz'
10	1R2
11	0R2
20	1R3
21	1R2
30	1L3
31	0L4
40	1R5
41	1L4
50	1LH
51	1R1

79 in 4948



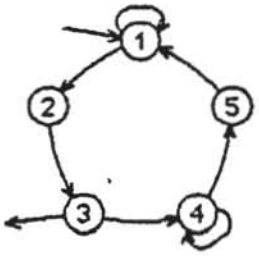
zr	wmz'
10	1R2
11	0R2
20	1L3
21	1R2
30	1LH
31	0L4
40	1R5
41	0L5
50	1R5
51	1R1

78 in 6671



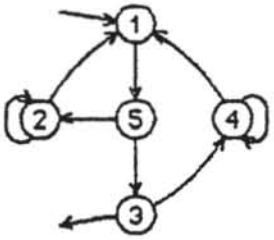
zr	wmz'
10	1R2
11	1L1
20	1L1
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	1L1
51	0L1

78 in 7036



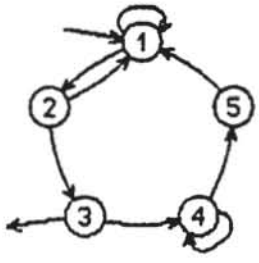
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	1L1
51	0L1

78 in 7046



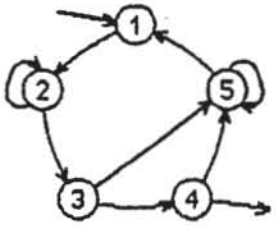
zr	wmz'
10	1R5
11	0L5
20	1L1
21	0L2
30	0R4
31	1LH
40	1R1
41	0L4
50	1L2
51	1R3

78 in 9108



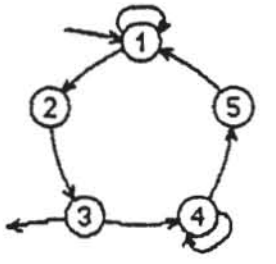
zr	wmz'
10	1R2
11	1L1
20	1L1
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	1R1
51	0L1

78 in 9238



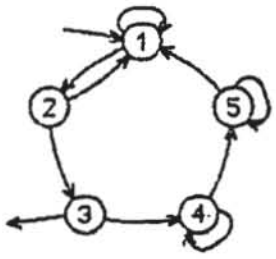
zr	wmz'
10	1R2
11	0L2
20	1R3
21	1L2
30	0R5
31	0R4
40	1LH
41	0R5
50	1L5
51	1L1

78 in 9255



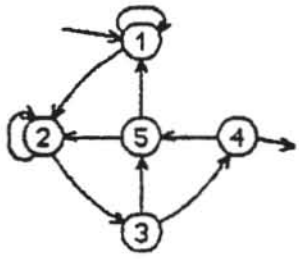
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	1R1
51	0L1

78 in 9258



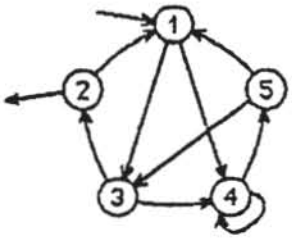
zr	wmz'
10	1R2
11	1L1
20	1L1
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	0R5
51	0L1

78 in 11440



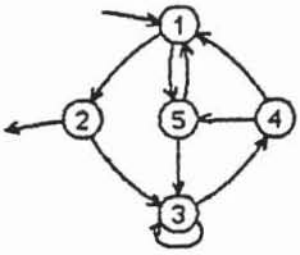
zr	wmz'
10	1R2
11	0R1
20	1R3
21	0L2
30	1L5
31	1R4
40	1LH
41	0R5
50	1R1
51	1L2

76 in 4172



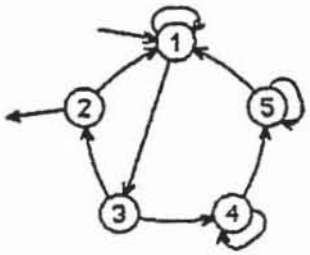
zr	wmz'
10	1R3
11	0R4
20	1LH
21	1R1
30	1L4
31	1R2
40	0L5
41	1R4
50	1R1
51	0R3

76 in 4359



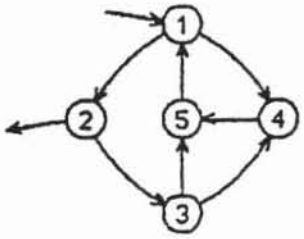
zr	wmz'
10	1R2
11	1L5
20	0R3
21	1LH
30	1R4
31	0L3
40	1L1
41	0R5
50	1R1
51	1L3

73 in 2052



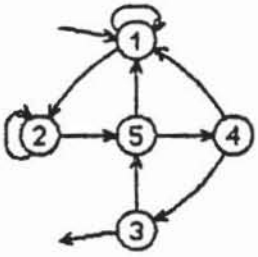
zr	wmz'
10	1R3
11	1R1
20	0R1
21	1LH
30	0R4
31	1R2
40	1L4
41	1R5
50	1L1
51	0L5

73 in 5167



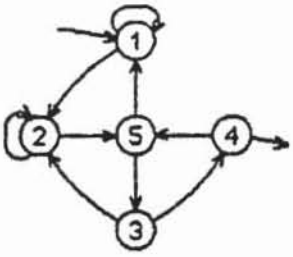
zr	wmz'
10	1R2
11	0L4
20	0R3
21	1LH
30	1L5
31	0R4
40	1L5
41	0R5
50	0L1
51	0R1

72 in 7383



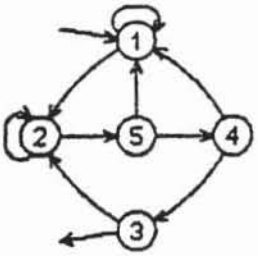
zr	wmz'
10	1R2
11	1R1
20	1L5
21	1L2
30	1LH
31	1L5
40	1L1
41	1L3
50	1R1
51	0L4

71 in 3545



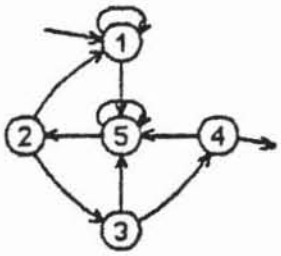
zr	wmz'
10	1R2
11	1R1
20	1L5
21	1L2
30	0L2
31	1L4
40	1LH
41	1L5
50	1R1
51	0L3

71 in 3549



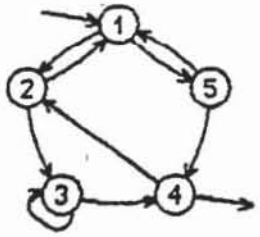
zr	wmz'
10	1R2
11	1R1
20	1L5
21	1L2
30	1LH
31	0R2
40	1L1
41	1L3
50	1R1
51	0L4

71 in 3691



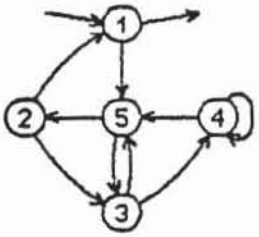
zr	wmz'
10	1R5
11	1R1
20	1R1
21	0L3
30	0L5
31	1L4
40	1LH
41	0R5
50	1L2
51	1L5

71 in 3699



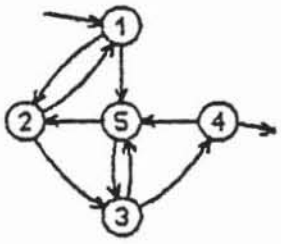
zr	wmz'
10	1R2
11	0L5
20	1L3
21	1R1
30	1L4
31	1L3
40	0R2
41	1LH
50	1R4
51	1L1

71 in 3902



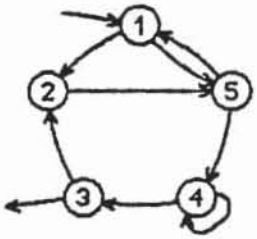
zr	wmz'
10	1R5
11	1LH
20	1R3
21	0R1
30	1L5
31	1L4
40	0L5
41	0R4
50	1R2
51	0L3

71 in 5988



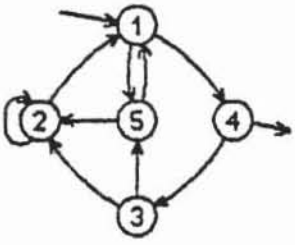
zr	wmz'
10	1R5
11	1R2
20	1R1
21	1L3
30	0R4
31	0L5
40	1LH
41	0L5
50	1L2
51	1R3

70 in 2821



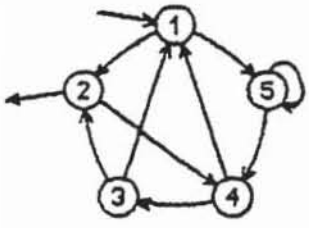
zr	wmz'
10	1R2
11	1R5
20	1L5
21	1L5
30	1LH
31	0L2
40	1L3
41	1R4
50	1R1
51	0R4

70 in 4916



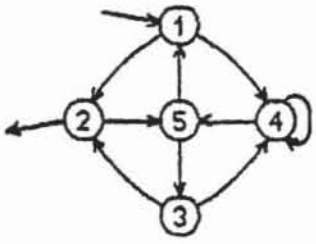
zr	wmz'
10	1R5
11	1R4
20	1L1
21	0L2
30	1L5
31	0R2
40	1LH
41	0R3
50	1L2
51	0R1

69 in 6956



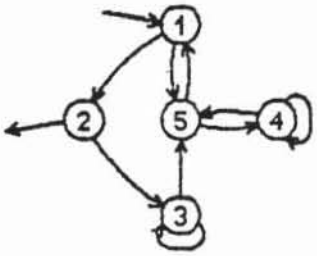
zr	wmz'
10	1R2
11	0L5
20	0R4
21	1LH
30	1L1
31	1R2
40	1R3
41	0L1
50	1L4
51	0L5

68 in 3630



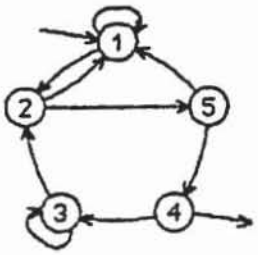
zr	wmz'
10	1R2
11	0L4
20	0R5
21	1LH
30	1L4
31	1R2
40	1L5
41	0L4
50	1R3
51	0L1

68 in 3630



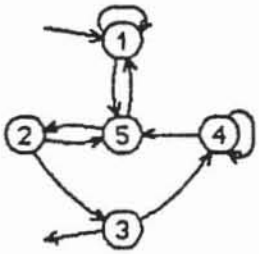
zr	wmz'
10	1R5
11	1L2
20	0L3
21	1LH
30	0L5
31	1L3
40	1R5
41	0R4
50	1L1
51	0R4

68 in 3737



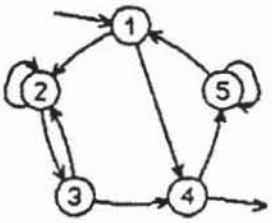
zr	wmz'
10	1R2
11	0R1
20	1L5
21	0R1
30	0L2
31	1L3
40	0L3
41	1LH
50	1R1
51	1L4

68 in 3737



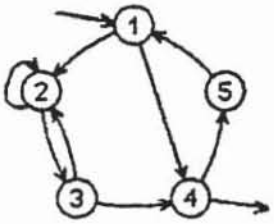
zr	wmz'
10	1R5
11	0R1
20	1R5
21	1L3
30	0L4
31	1LH
40	0L5
41	1L4
50	1L2
51	0R1

68 in 3737



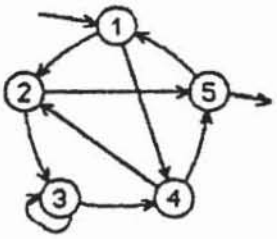
zr	wmz'
10	1R2
11	1L4
20	0R3
21	0R2
30	0L4
31	1R2
40	1L5
41	1LH
50	0L1
51	1L5

67 in 3858



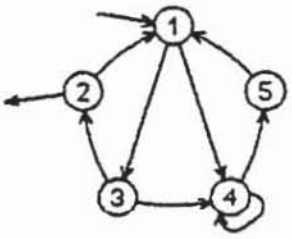
zr	wmz'
10	1R2
11	1L4
20	0R3
21	0R2
30	0L4
31	1R2
40	1L5
41	1LH
50	0L1
51	0R1

67 in 4128



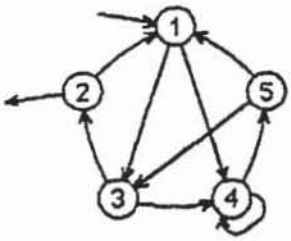
zr	wmz'
10	1R2
11	0L4
20	1L3
21	1R5
30	0L4
31	0L3
40	1R5
41	1R2
50	0R1
51	1LH

67 in 4598



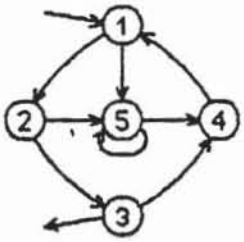
zr	wmz'
10	1R3
11	1L4
20	0R1
21	1LH
30	1L4
31	1R2
40	1R5
41	0L4
50	0L1
51	0R1

67 in 4682



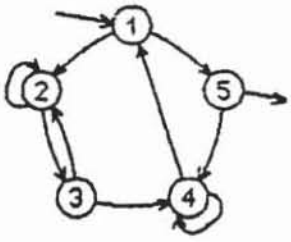
zr	wmz'
10	1R3
11	1L4
20	0R1
21	1LH
30	1L4
31	1R2
40	1R5
41	0L4
50	0L1
51	0L3

67 in 4868



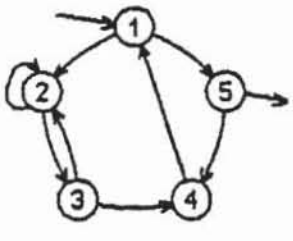
zr	wmz'
10	1R2
11	1L5
20	1L5
21	1R3
30	1L4
31	1LH
40	0L1
41	0R1
50	1R4
51	0L5

67 in 6742



zr	wmz'
10	1R2
11	1L5
20	0R3
21	0R2
30	1L4
31	1R2
40	0L1
41	1L4
50	1L4
51	1LH

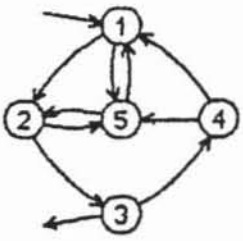
66 in 3355



zr	wmz'
10	1R2
11	1L5
20	0R3
21	0R2
30	1L4
31	1R2
40	0L1
41	0R1
50	1L4
51	1LH

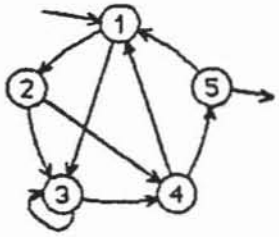
66 in 3457

Appendix B: Examples of TMs, which make a large number of steps



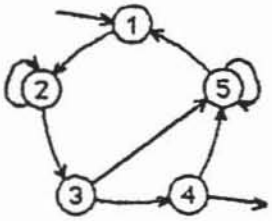
zr	wmz'
10	1R2
11	0L5
20	1R5
21	1R3
30	0R4
31	1LH
40	1L5
41	1R1
50	1L1
51	0R2

501 in 134467



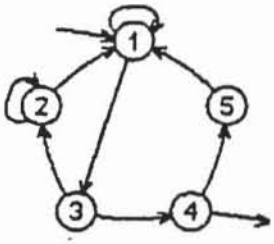
zr	wmz'
10	1R2
11	0R3
20	1L3
21	0R4
30	0L4
31	0L3
40	1R5
41	0L1
50	0R1
51	1LH

208 in 49249



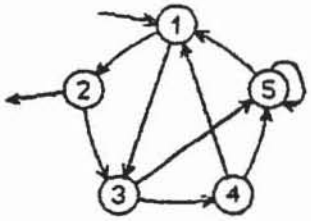
zr	wmz'
10	1R2
11	0L2
20	1R3
21	1L2
30	1L5
31	0R4
40	1LH
41	0R5
50	1L5
51	1L1

162 in 41710



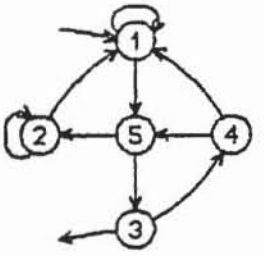
zr	wmz'
10	1R3
11	1R1
20	1L1
21	1L2
30	1L2
31	0R4
40	1LH
41	1R5
50	0R1
51	0L1

240 in 41360



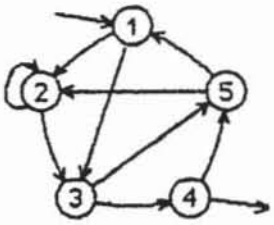
zr	wmz'
10	1R2
11	0L3
20	0R3
21	1LH
30	1R4
31	0R5
40	1L5
41	0R1
50	0L1
51	0L5

216 in 40899



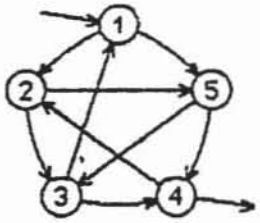
zr	wmz'
10	1R5
11	1R1
20	1L1
21	1L2
30	1LH
31	1R4
40	0R1
41	1R5
50	1L2
51	0R3

240 in 40806



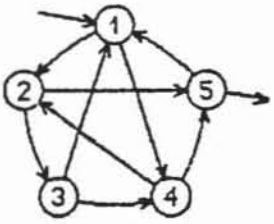
zr	wmz'
10	1R2
11	0L3
20	0R3
21	0R2
30	1L4
31	0R5
40	0L5
41	1LH
50	1L1
51	0L2

208 in 39734



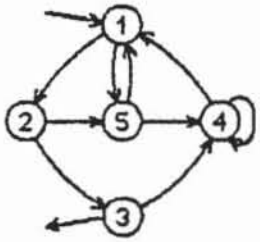
zr	wmz'
10	1R2
11	1L5
20	1L3
21	1R5
30	1L1
31	0R4
40	1LH
41	0L2
50	1L4
51	0R3

157 in 33178



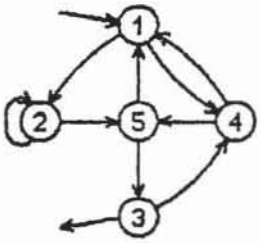
zr	wmz'
10	1R2
11	1L4
20	1R3
21	0L5
30	1L1
31	1R4
40	1R5
41	0L2
50	1LH
51	0R1

157 in 33174



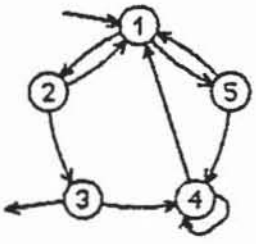
zr	wmz'
10	1R2
11	0L5
20	1L5
21	1R3
30	0R4
31	1LH
40	0R1
41	1R4
50	1L1
51	1L4

125 in 24778



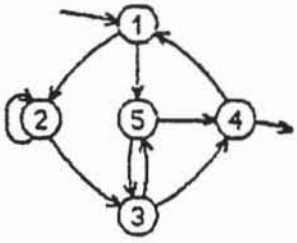
zr	wmz'
10	1R2
11	1L4
20	1R5
21	0R2
30	1LH
31	0R4
40	1L5
41	0L1
50	1L1
51	1L3

36 in 24567



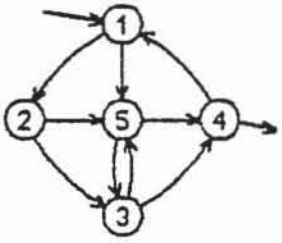
zr	wmz'
10	1R2
11	0L5
20	1L1
21	1R3
30	0R4
31	1LH
40	0R1
41	1R4
50	1L1
51	1L4

125 in 23838



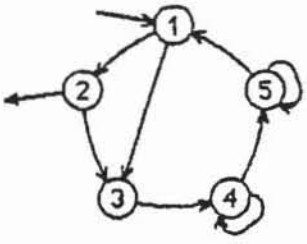
zr	wmz'
10	1R2
11	1L5
20	0L2
21	1R3
30	1L5
31	0L4
40	1LH
41	0R1
50	1R4
51	1L3

161 in 22335



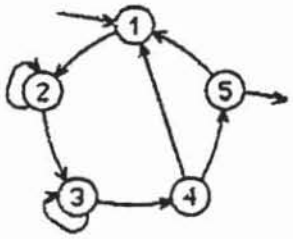
zr	wmz'
10	1R2
11	1L5
20	1L5
21	1R3
30	1L5
31	0L4
40	1LH
41	0R1
50	1R4
51	1L3

161 in 22059



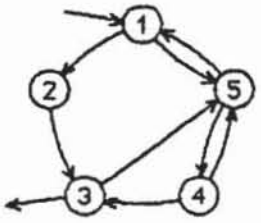
zr	wmz'
10	1R3
11	1L2
20	1LH
21	0L3
30	1L4
31	0R4
40	1R5
41	1R4
50	0L1
51	1L5

168 in 21294



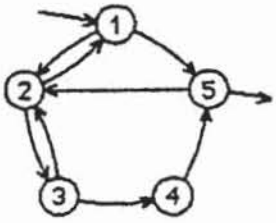
zr	wmz'
10	1R2
11	0L2
20	1L3
21	1L2
30	0R4
31	1R3
40	1L1
41	1R5
50	1LH
51	0R1

168 in 21286



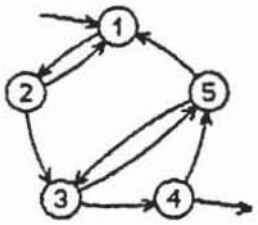
zr	wmz'
10	1R2
11	1R5
20	1L3
21	0L3
30	1LH
31	0R5
40	1L5
41	0L3
50	1R1
51	0L4

80 in 21044



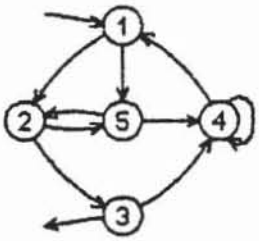
zr	wmz'
10	1R2
11	0R5
20	1L3
21	0R1
30	1L4
31	1L2
40	1R5
41	0R5
50	1LH
51	0L2

80 in 21040



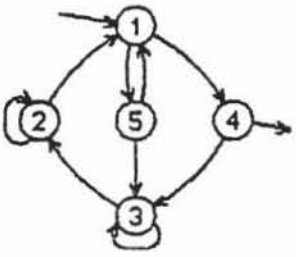
zr	wmz'
10	1R2
11	1L2
20	1L1
21	0R3
30	0L5
31	1R4
40	1LH
41	0L5
50	0R1
51	1L3

81 in 21010



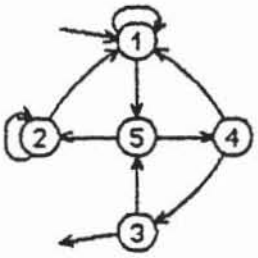
zr	wmz'
10	1R2
11	0R5
20	1R5
21	1R3
30	1LH
31	0R4
40	0L1
41	1R4
50	1L4
51	1R2

161 in 20796



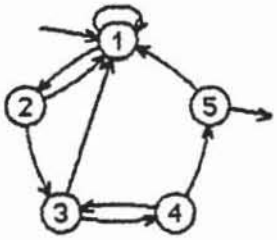
zr	wmz'
10	1R5
11	0R4
20	0R2
21	1L1
30	1L2
31	1L3
40	1LH
41	0L3
50	1L3
51	1R1

159 in 20766



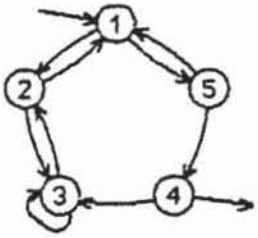
zr	wmz'
10	1R5
11	0L1
20	1L2
21	1L1
30	1LH
31	0R5
40	1R1
41	1R3
50	0R2
51	1R4

154 in 20739



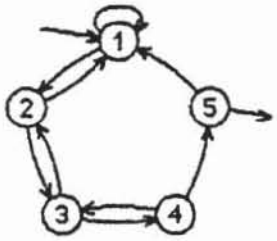
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R1
30	0R1
31	1L4
40	1L3
41	0L5
50	1LH
51	0L1

161 in 20729



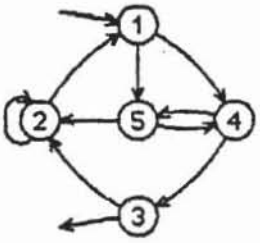
zr	wmz'
10	1R2
11	1L5
20	1L1
21	0R3
30	1R2
31	1L3
40	1LH
41	0L3
50	1L1
51	0L4

161 in 20711



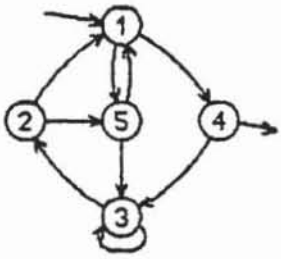
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R1
30	1R2
31	1L4
40	1L3
41	0L5
50	1LH
51	0L1

161 in 20711



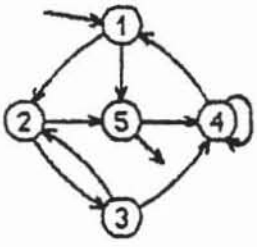
zr	wmz'
10	1R5
11	1L4
20	1L1
21	1L2
30	1LH
31	0L2
40	1R5
41	0R3
50	1L2
51	1R4

159 in 20492



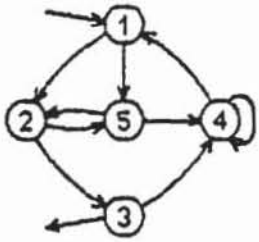
zr	wmz'
10	1R5
11	0R4
20	1R5
21	1L1
30	1L2
31	1L3
40	1LH
41	0L3
50	1L3
51	1R1

159 in 20492



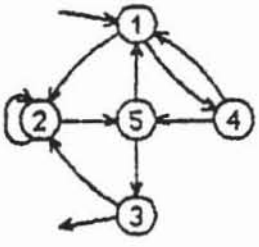
zr	wmz'
10	1R2
11	1L5
20	1R3
21	1R5
30	1L4
31	1R2
40	0L1
41	1L4
50	1LH
51	0R4

161 in 20398



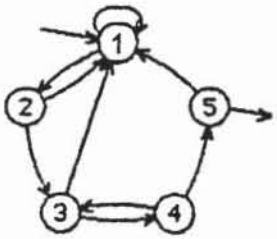
zr	wmz'
10	1R2
11	0L5
20	1L5
21	0L3
30	1LH
31	0L4
40	0R1
41	1L4
50	1R4
51	1L2

157 in 19463



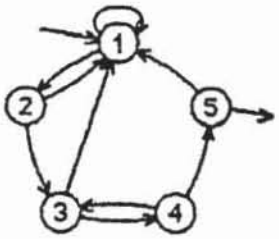
zr	wmz'
10	1R2
11	1L4
20	1R5
21	0R2
30	1LH
31	1R2
40	1L5
41	0L1
50	1L1
51	0L3

35 in 19283



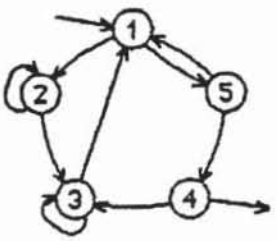
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R1
30	1R1
31	1L4
40	1L3
41	0L5
50	1LH
51	0L1

157 in 19145



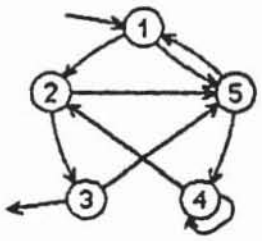
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R1
30	1L1
31	1L4
40	1L3
41	0L5
50	1LH
51	0L1

157 in 19129



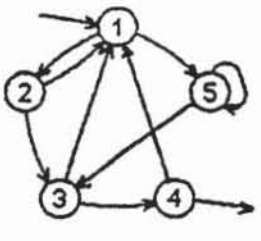
zr	wmz'
10	1R2
11	0L5
20	0L3
21	0R2
30	1R1
31	1L3
40	0L3
41	1LH
50	0L4
51	1L1

99 in 18040



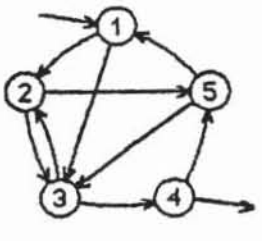
zr	wmz'
10	1R2
11	1R5
20	0L3
21	0L5
30	1LH
31	1L5
40	1L2
41	1R4
50	1R1
51	0R4

83 in 16351



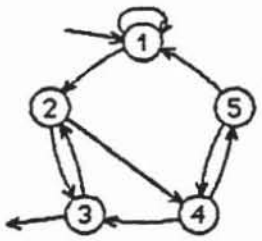
zr	wmz'
10	1R2
11	0R5
20	1R3
21	1R1
30	0L4
31	0L1
40	1LH
41	1L1
50	1L3
51	1R5

83 in 16328



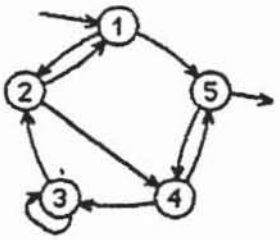
zr	wmz'
10	1R3
11	1R2
20	1L5
21	1R3
30	0R2
31	0L4
40	1LH
41	0L5
50	1R1
51	1L3

43 in 15889



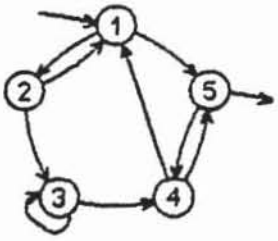
zr	wmz'
10	1R2
11	1L1
20	1R4
21	1L3
30	1LH
31	0L2
40	1R5
41	1R3
50	0L1
51	1R4

165 in 15589



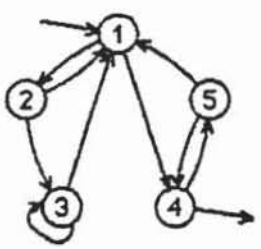
zr	wmz'
10	1R2
11	1R5
20	1R4
21	1R1
30	0R2
31	1L3
40	1L3
41	1L5
50	1LH
51	0L4

163 in 15271



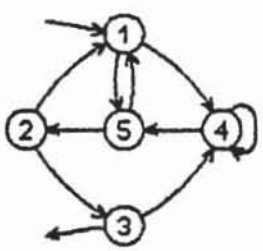
zr	wmz'
10	1R2
11	1R5
20	0L3
21	1R1
30	1L4
31	1L3
40	1R1
41	1L5
50	1LH
51	0L4

163 in 14975



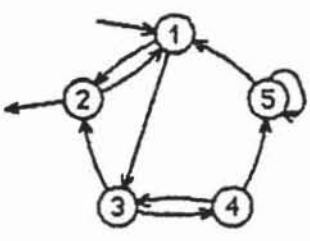
zr	wmz'
10	1R2
11	1R4
20	1L3
21	1R1
30	1R1
31	1L3
40	1LH
41	0L5
50	1R1
51	1L4

163 in 14965



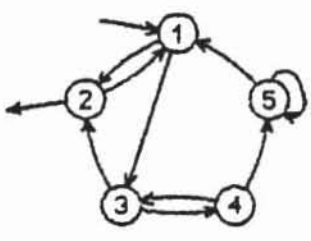
zr	wmz'
10	1R5
11	1R4
20	1R1
21	1L3
30	0L4
31	1LH
40	0L5
41	1L4
50	1L2
51	0R1

97 in 14661



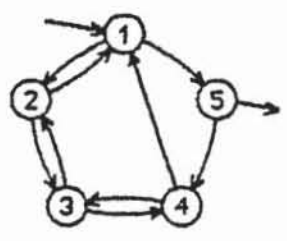
zr	wmz'
10	1R3
11	1L2
20	1LH
21	0L1
30	1R4
31	1R2
40	1L5
41	1R3
50	1R1
51	1L5

161 in 14389



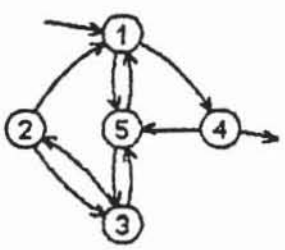
zr	wmz'
10	1R3
11	1L2
20	1LH
21	0L1
30	1R4
31	1R2
40	1L5
41	1R3
50	0L1
51	1L5

161 in 14371



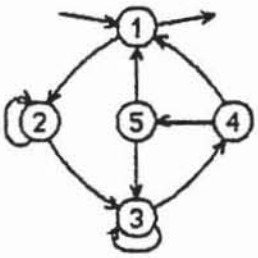
zr	wmz'
10	1R2
11	1R5
20	1L3
21	0R1
30	0L4
31	1L2
40	0R1
41	1L3
50	1LH
51	0L4

81 in 14370



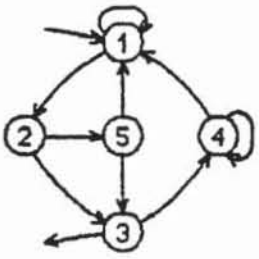
zr	wmz'
10	1R5
11	0R4
20	0R1
21	1L3
30	0L2
31	1L5
40	1LH
41	0L5
50	1L3
51	0R1

80 in 14370



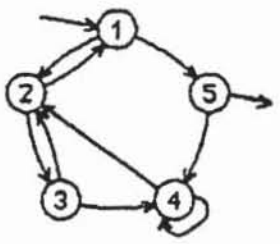
zr	wmz'
10	1R2
11	1LH
20	1L3
21	0R2
30	1L4
31	0R3
40	0L5
41	1L1
50	0R1
51	0L3

83 in 14268



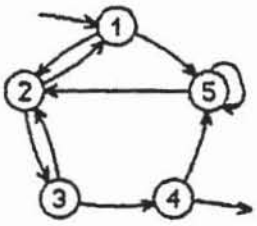
zr	wmz'
10	1R2
11	0L1
20	0R5
21	1R3
30	1L4
31	1LH
40	1R1
41	0L4
50	0L3
51	0R1

83 in 14264



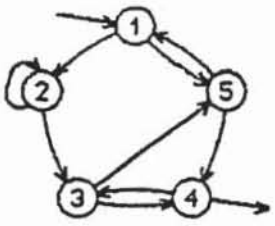
zr	wmz'
10	1R2
11	1L5
20	1L1
21	0R3
30	1R2
31	1R4
40	0L2
41	1L4
50	0L4
51	1LH

97 in 14095



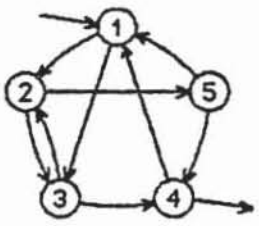
zr	wmz'
10	1R2
11	1R5
20	1L3
21	0R1
30	1R2
31	1L4
40	0L5
41	1LH
50	0L2
51	1L5

97 in 14095



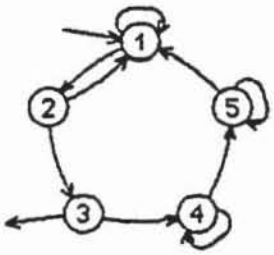
zr	wmz'
10	1R2
11	1L5
20	1R3
21	1R2
30	1L5
31	1R4
40	1LH
41	0R3
50	1L1
51	1L4

159 in 13821



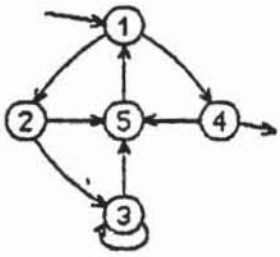
zr	wmz'
10	1R2
11	0L3
20	1R5
21	1R3
30	0R2
31	0L4
40	1LH
41	1L1
50	1L4
51	1R1

41 in 12045



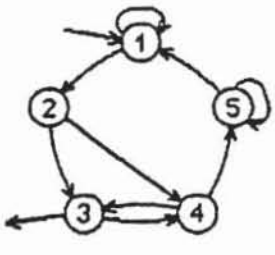
zr	wmz'
10	1R2
11	1L1
20	1L1
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	0R5
51	0L1

78 in 11440



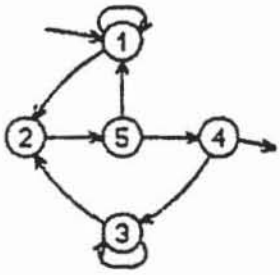
zr	wmz'
10	1R2
11	0L4
20	1R5
21	1R3
30	0R5
31	0L3
40	1LH
41	1L5
50	1L1
51	0L1

41 in 11384



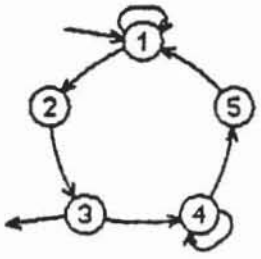
zr	wmz'
10	1R2
11	1R1
20	1R4
21	1L3
30	1LH
31	0R4
40	1L5
41	1R3
50	1L1
51	1L5

105 in 10707



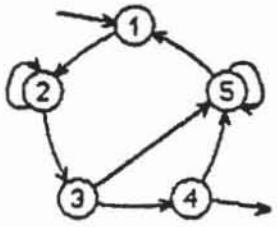
zr	wmz'
10	1R2
11	0R1
20	1L5
21	0R5
30	1L2
31	0R3
40	0L3
41	1LH
50	1R1
51	1L4

81 in 9865



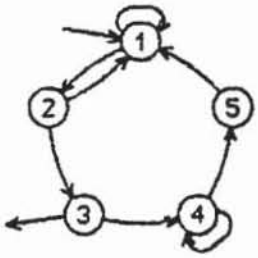
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	1R1
51	0L1

78 in 9258



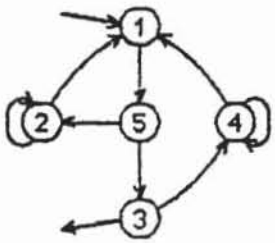
zr	wmz'
10	1R2
11	0L2
20	1R3
21	1L2
30	0R5
31	0R4
40	1LH
41	0R5
50	1L5
51	1L1

78 in 9255



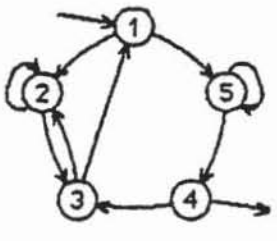
zr	wmz'
10	1R2
11	1L1
20	1L1
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	1R1
51	0L1

78 in 9238



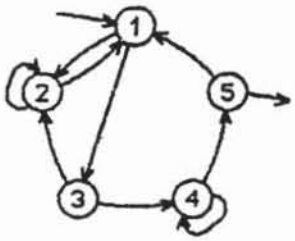
zr	wmz'
10	1R5
11	0L5
20	1L1
21	0L2
30	0R4
31	1LH
40	1R1
41	0L4
50	1L2
51	1R3

78 in 9108



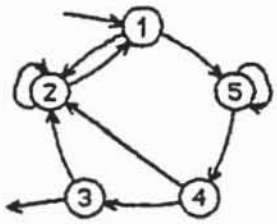
zr	wmz'
10	1R2
11	0R5
20	1L3
21	0L2
30	1R1
31	0L2
40	0R3
41	1LH
50	1R5
51	1R4

91 in 8245



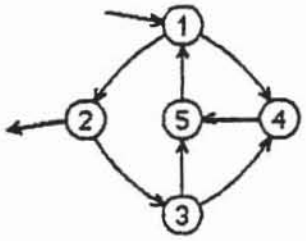
zr	wmz'
10	1R3
11	0L2
20	1L1
21	0L2
30	1R2
31	0R4
40	1R4
41	1R5
50	0R1
51	1LH

88 in 7706



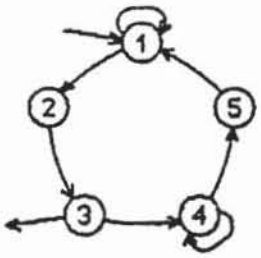
zr	wmz'
10	1R2
11	0L5
20	1L1
21	1R2
30	1LH
31	0R2
40	0L2
41	1R3
50	1R5
51	1R4

81 in 7682



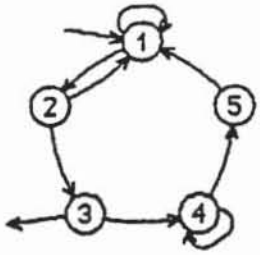
zr	wmz
10	1R2
11	0L4
20	0R3
21	1LH
30	1L5
31	0R4
40	1L5
41	0R5
50	0L1
51	0R1

72 in 7383



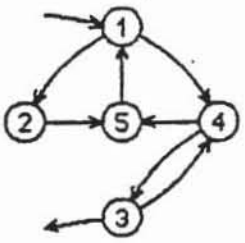
zr	wmz'
10	1R2
11	1L1
20	1L3
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	1L1
51	0L1

78 in 7046



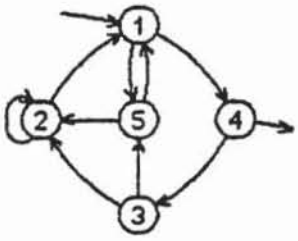
zr	wmz'
10	1R2
11	1L1
20	1L1
21	0R3
30	1LH
31	0R4
40	1L4
41	1L5
50	1L1
51	0L1

78 in 7036



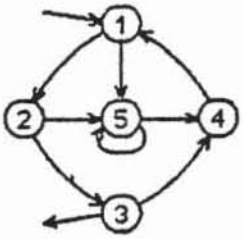
zr	wmz'
10	1R2
11	1R4
20	0R5
21	1R5
30	1LH
31	0L4
40	1L5
41	1L3
50	1L1
51	0L1

36 in 6987



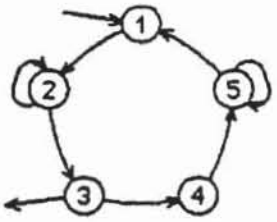
zr	wmz'
10	1R5
11	1R4
20	1L1
21	0L2
30	1L5
31	0R2
40	1LH
41	0R3
50	1L2
51	0R1

69 in 6956



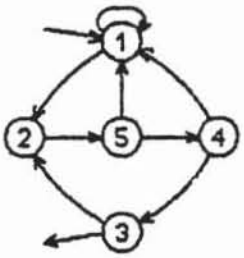
zr	wmz'
10	1R2
11	1L5
20	1L5
21	1R3
30	1L4
31	1LH
40	0L1
41	0R1
50	1R4
51	0L5

67 in 6742



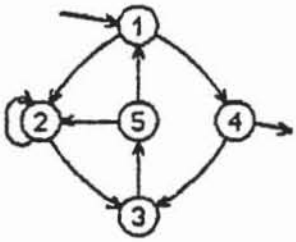
zr	wmz'
10	1R2
11	0R2
20	1L3
21	1R2
30	1LH
31	0L4
40	1R5
41	0L5
50	1R5
51	1R1

78 in 6671



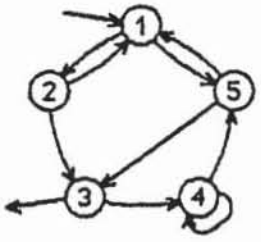
zr	wmz'
10	1R2
11	0L1
20	0L5
21	0R5
30	1L2
31	1LH
40	1L1
41	1R3
50	1R4
51	1L1

65 in 6240



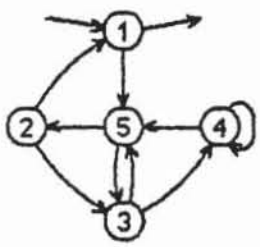
zr	wmz'
10	1R2
11	1L4
20	1L3
21	0R2
30	0R5
31	0L5
40	1R3
41	1LH
50	1L1
51	1R2

65 in 6238



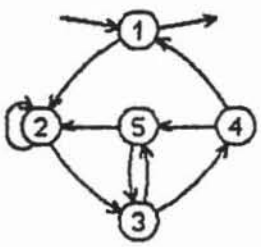
zr	wmz'
10	1R2
11	0L5
20	1R3
21	0R1
30	1L4
31	1LH
40	1L5
41	1L4
50	1L1
51	0L3

112 in 6147



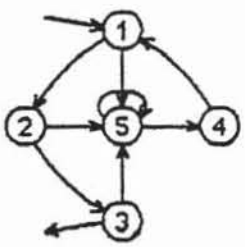
zr	wmz'
10	1R5
11	1LH
20	1R3
21	0R1
30	1L5
31	1L4
40	0L5
41	0R4
50	1R2
51	0L3

71 in 5988



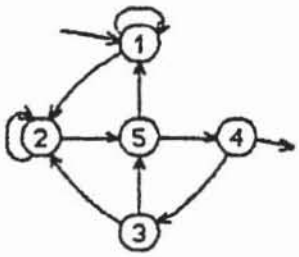
zr	wmz'
10	1R2
11	1LH
20	0R3
21	0R2
30	1R4
31	0L5
40	1R5
41	0R1
50	1L3
51	1L2

59 in 5940



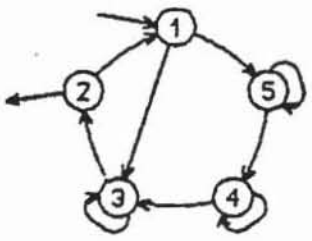
zr	wmz'
10	1R2
11	1L5
20	1L5
21	0R3
30	1L5
31	1LH
40	0L1
41	0R1
50	1R4
51	0L5

66 in 5712



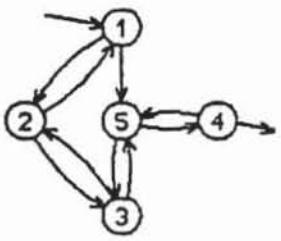
zr	wmz'
10	1R2
11	0R1
20	1L5
21	0L2
30	0L5
31	1L2
40	1L3
41	1LH
50	1R1
51	1L4

89 in 5685



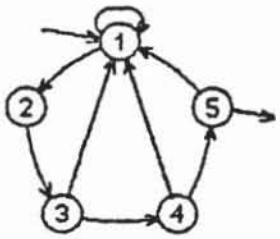
zr	wmz'
10	1R3
11	0L5
20	1LH
21	1L1
30	1L2
31	1R3
40	1R4
41	0R3
50	1L4
51	1L5

81 in 5323



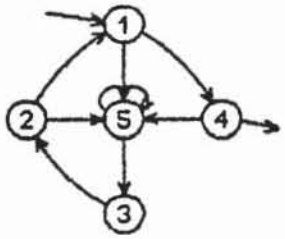
zr	wmz'
10	1R5
11	1L2
20	1R3
21	0L1
30	1L5
31	1R2
40	1LH
41	1L5
50	0R3
51	1L4

83 in 5309



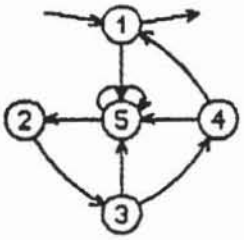
zr	wmz'
10	1R2
11	0L1
20	0L3
21	0R3
30	1R4
31	1L1
40	1L1
41	0R5
50	1L1
51	1LH

64 in 5294



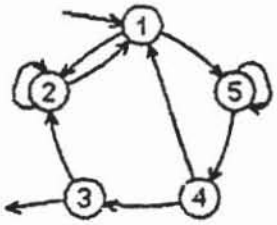
zr	wmz'
10	1R5
11	0L4
20	1L1
21	1R5
30	0R2
31	0L2
40	1R5
41	1LH
50	1L3
51	0R5

64 in 5292



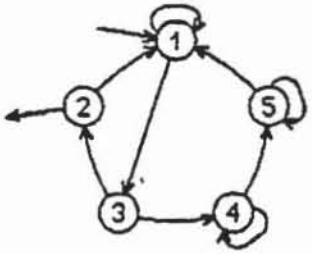
zr	wmz'
10	1R5
11	1LH
20	0R3
21	0L3
30	1L4
31	1R5
40	1R5
41	0L1
50	1L2
51	0R5

64 in 5292



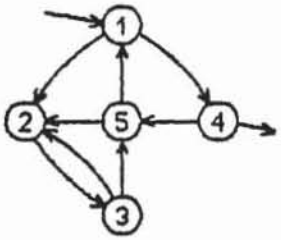
zr	wmz'
10	1R2
11	0L5
20	1L1
21	1R2
30	1LH
31	0R2
40	1L1
41	1R3
50	1R5
51	1R4

81 in 5230



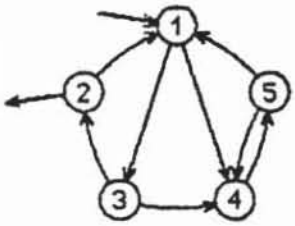
zr	wmz'
10	1R3
11	1R1
20	0R1
21	1LH
30	0R4
31	1R2
40	1L4
41	1R5
50	1L1
51	0L5

73 in 5167



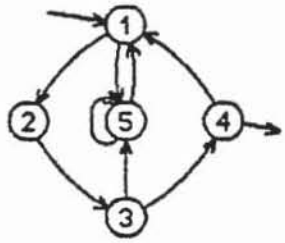
zr	wmz'
10	1R2
11	0L4
20	1L3
21	0R3
30	1R5
31	1R2
40	1L5
41	1LH
50	1L1
51	0L2

64 in 5165



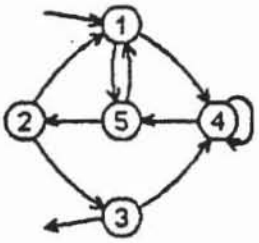
zr	wmz'
10	1R3
11	0R4
20	1R1
21	1LH
30	1L4
31	0R2
40	1R5
41	0L5
50	1L1
51	1L4

64 in 5164



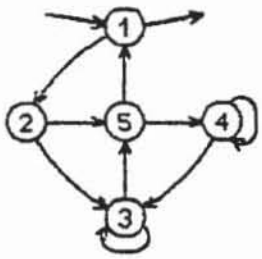
zr	wmz'
10	1R2
11	0L5
20	0R3
21	0L3
30	1R5
31	0R4
40	1R1
41	1LH
50	1L1
51	0L5

59 in 5108



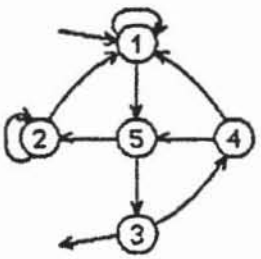
zr	wmz'
10	1R5
11	1R4
20	1L1
21	0L3
30	1L4
31	1LH
40	0L5
41	0L4
50	1L2
51	0R1

53 in 5097



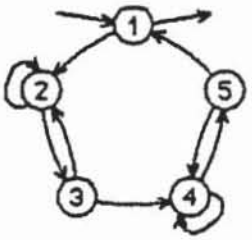
zr	wmz'
10	1R2
11	1LH
20	0R5
21	1R3
30	1R5
31	0R3
40	1L3
41	0L4
50	1L4
51	1R1

83 in 5074



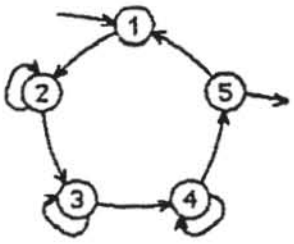
zr	wmz'
10	1R5
11	0R1
20	1L1
21	0L2
30	1R4
31	1LH
40	0R5
41	1R1
50	1L2
51	1R3

83 in 5067



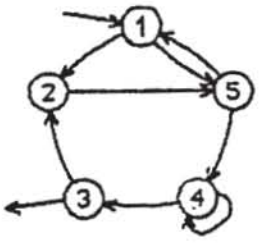
zr	wmz'
10	1R2
11	1LH
20	1L2
21	0R3
30	0R2
31	1L4
40	0L5
41	1L4
50	1L1
51	0L4

106 in 5012



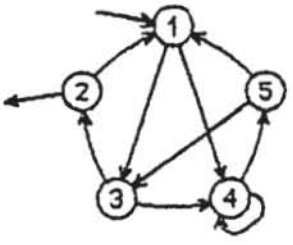
zr	wmz'
10	1R2
11	0R2
20	1R3
21	1R2
30	1L3
31	0L4
40	1R5
41	1L4
50	1LH
51	1R1

79 in 4948



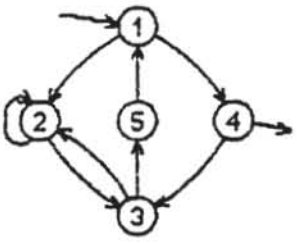
zr	wmz'
10	1R2
11	1R5
20	1L5
21	1L5
30	1LH
31	0L2
40	1L3
41	1R4
50	1R1
51	0R4

70 in 4916



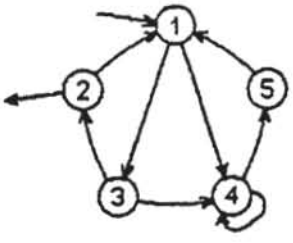
zr	wmz'
10	1R3
11	1L4
20	0R1
21	1LH
30	1L4
31	1R2
40	1R5
41	0L4
50	0L1
51	0L3

67 in 4868



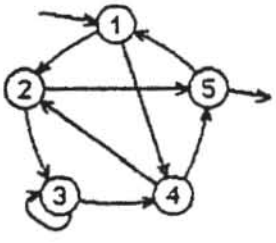
zr	wmz'
10	1R2
11	0R4
20	1L3
21	0L2
30	1R5
31	0L2
40	1R3
41	1LH
50	0R1
51	0L1

58 in 4836



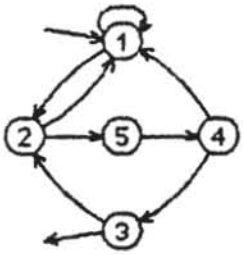
zr	wmz'
10	1R3
11	1L4
20	0R1
21	1LH
30	1L4
31	1R2
40	1R5
41	0L4
50	0L1
51	0R1

67 in 4682



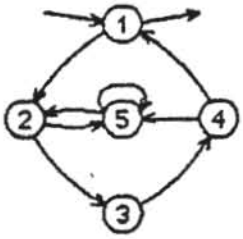
zr	wmz'
10	1R2
11	0L4
20	1L3
21	1R5
30	0L4
31	0L3
40	1R5
41	1R2
50	0R1
51	1LH

67 in 4598



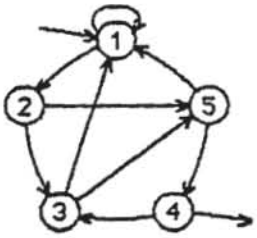
zr	wmz'
10	1R2
11	0R1
20	1L5
21	0R1
30	1L2
31	1LH
40	1L1
41	0L3
50	0L4
51	0R4

57 in 4565



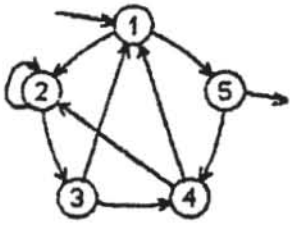
zr	wmz'
10	1R2
11	1LH
20	1R3
21	0L5
30	0R4
31	0L4
40	1R5
41	0R1
50	1L2
51	0L5

57 in 4561



zr	wmz'
10	1R2
11	0L1
20	0L3
21	0L5
30	1R5
31	1L1
40	0R3
41	1LH
50	1L1
51	1R4

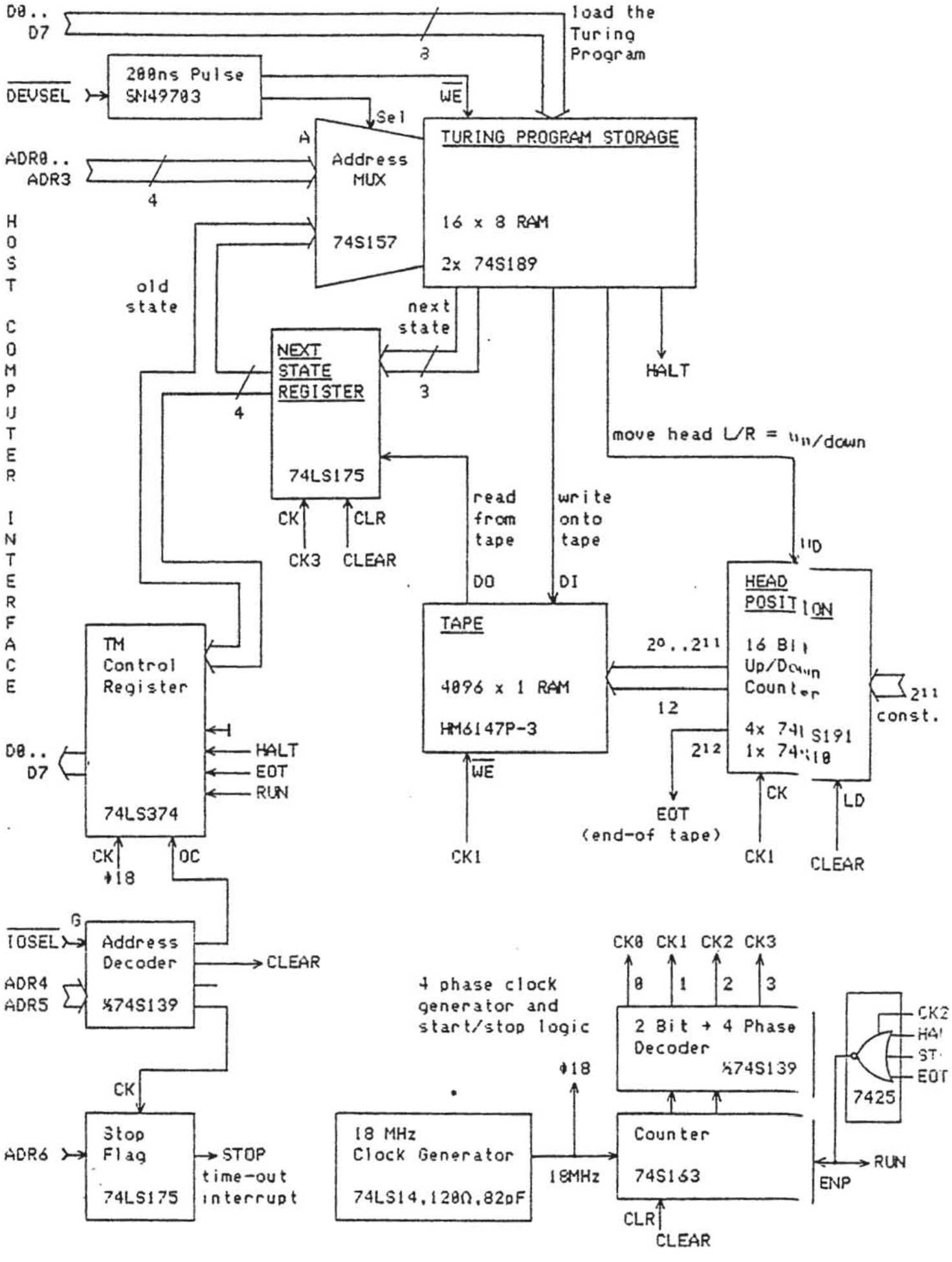
65 in 4534



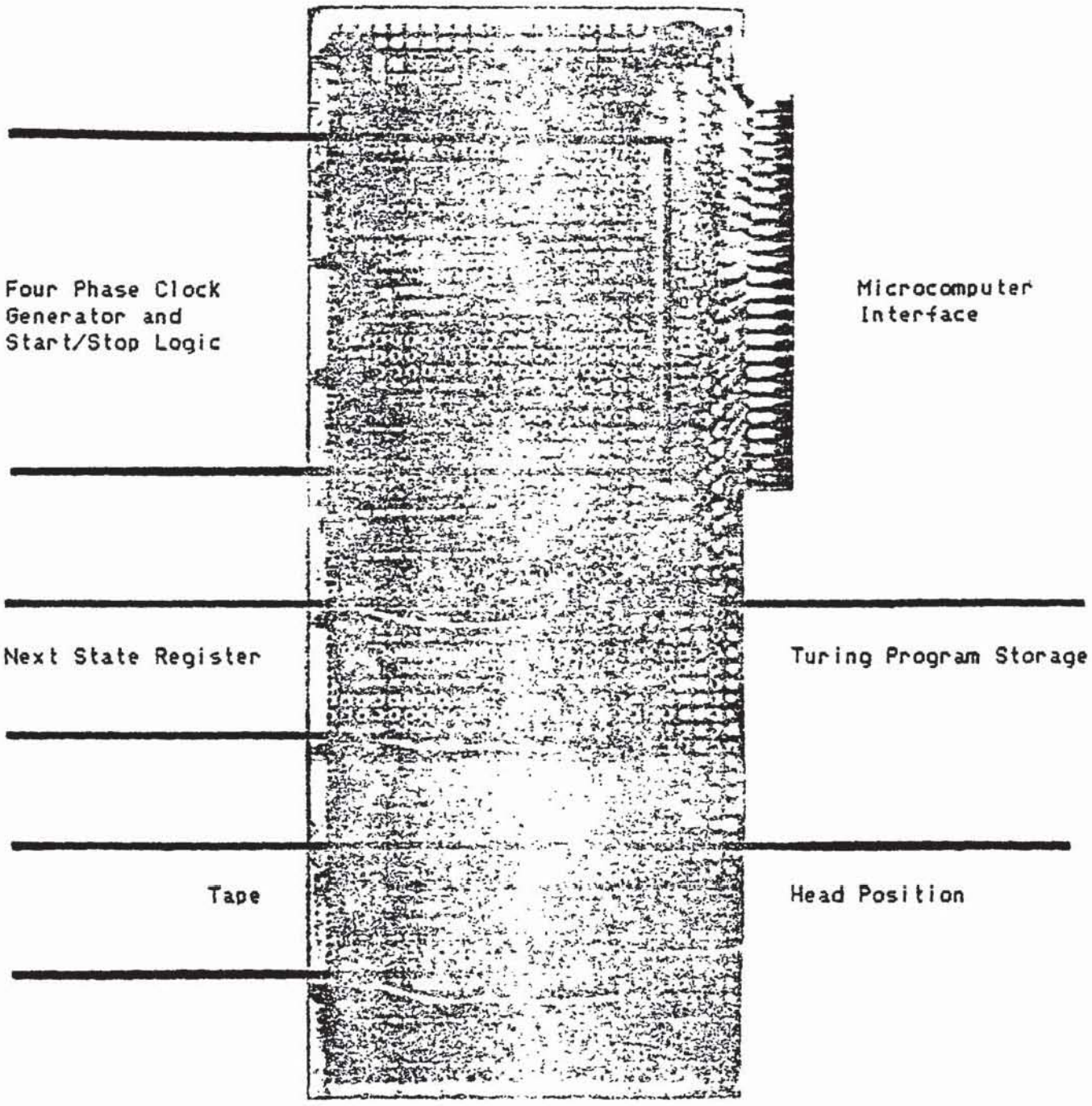
zr	wmz'
10	1R2
11	1L5
20	1L3
21	0R2
30	0R4
31	0R1
40	1L1
41	1R2
50	0L4
51	1LH

65 in 4532

Appendix C: Architecture of the Turing Machine Hardware



Appendix D: The Sections of the TM-Board

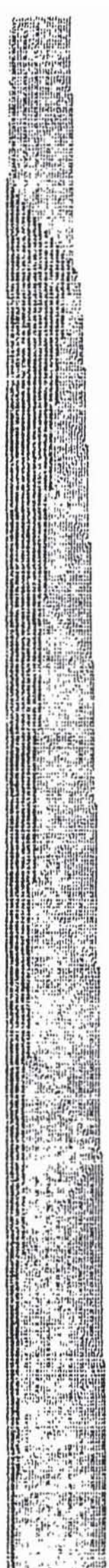


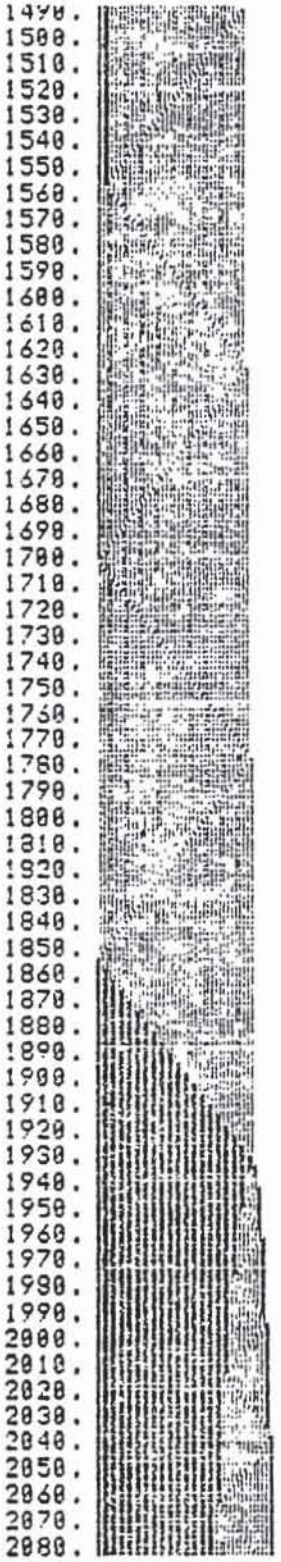
Appendix E: Output of the Busy-Beaver (Except)

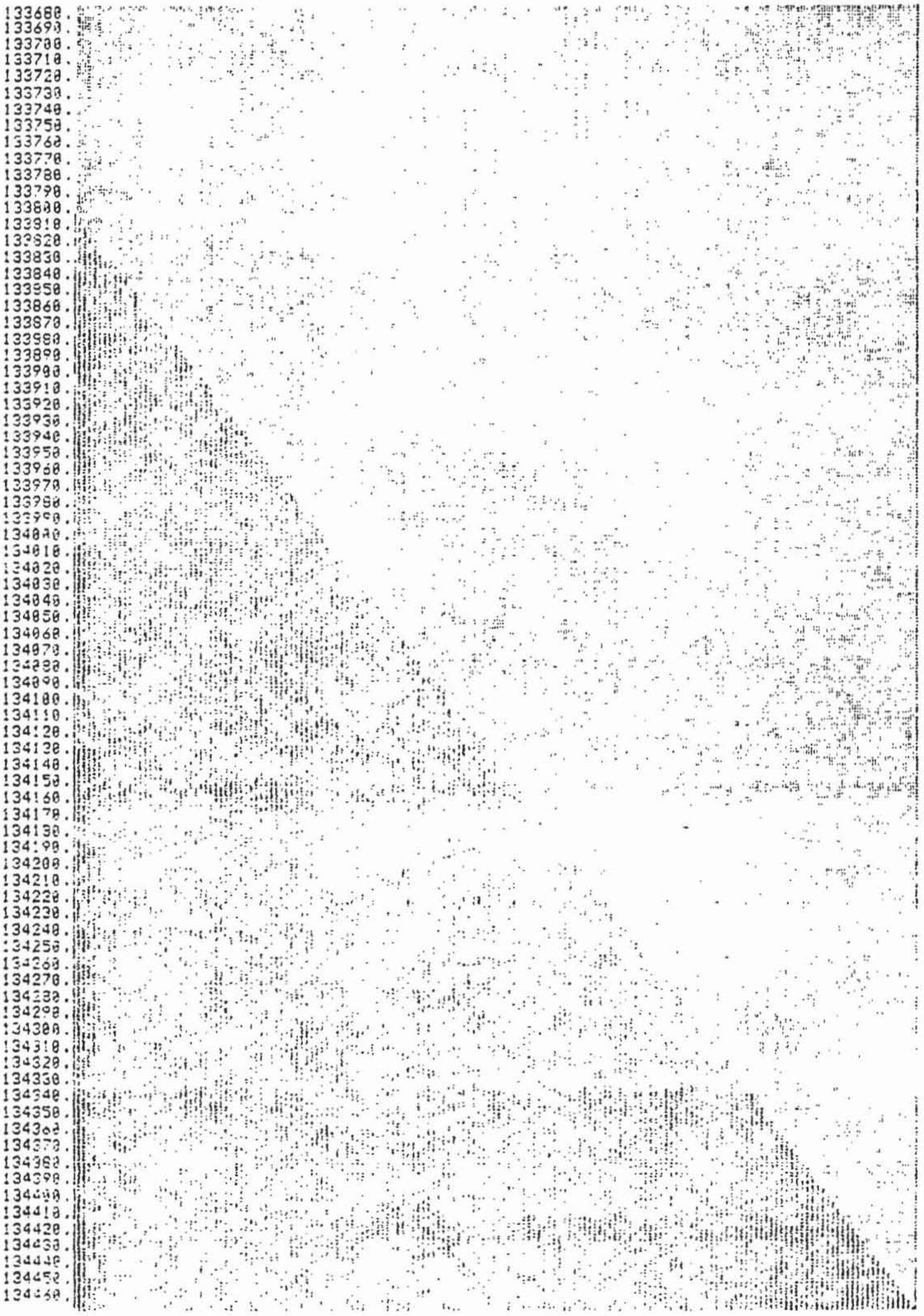
TM:

Z	R	W	M	Z'
0	0	1	R	1
0	1	0	L	2
1	0	1	R	2
1	1	1	R	3
2	0	1	L	0
2	1	0	R	1
3	0	0	R	4
3	1	1	L	F
4	0	1	L	2
4	1	1	R	0

START: Z=1







Literature

1) Turing, A.M. "On computable numbers, with an application to the Entscheidungsproblem". Proceedings of the London Mathematical Society (2), 42 (1936-1937), 230-265.

2) Kleene, S.C. "Introduction to Metamathematics". D. van Nostrand Co., Princeton, New Jersey, 1952.

3) Rado, T. "On non-computable functions". Bell System Technical Journal, 41, 3 (1962), 877-884.

4) Rado, T. "On a sample source for non-computable functions". Proceedings of the Symposium on Mathematical Theory of Automata, Brooklyn, New-York, 1963, 75-81.

5) Lin, S.; Rada, T. "Computer Studies of Turing Machine Problems". Journal of the ACM, 12 2 (1965), 196-212.

6) Weimann, B.; Casper, K.; Fenzl, W. "Untersuchung über haltende Programme für Turing-Maschinen mit 2 Zeichen und bis zu 5 Befehlen". GI-Jahrestatung 1972, Lecture Notes in Economics and Math. Systems No. 78 (1973), Springer Heidelberg,72-81.

7) Weimann, B. "Untersuchungen über Rado's Sigma-Funktion und eingeschränkte Halteprobleme bei Turingmaschinen". Dissertation, Rheinische Friedrich-Wilhelms-Universität Bonn, 1973.

8) Green, M.W. " A Lower Bound on Rado's Sigma Function for Binary Turing Machines". Proceedings of the 5th Annual Symposium on Switching Circuits Theory and Logical Design, Princeton, New Jersey, 1964, 91-94.

9) Cremers, A.B.; Kriegel, H.-P. (eds) "Theoretical Computer Science", 6th GI-Conference, Lecture Notes in Computer Science No. 145, Springer, Berlin-Heidelberg-New York, 1982.