

THE UNIVERSITY OF MICHIGAN
COLLEGE OF LITERATURE, SCIENCE, AND THE ARTS
Computer and Communication Sciences Department

DESTRUCTION OF A UNIVERSAL
COMPUTER-CONSTRUCTOR
IN CODD'S CELLULAR SPACE

Ulrich Golze

August 1972

Technical Report No. 137

with assistance from:

National Science Foundation
Grant No. GJ-29989X
Washington, D.C.

and the

Computational Facilities of
The Technical University of Hanover, Germany

THE UNIVERSITY OF MICHIGAN
ENGINEERING LIBRARY

21.7.1

UNK

1651

ABSTRACT

A universal self-reproducing computer can be embedded into the cellular space designed by E.F. Codd in CELLULAR AUTOMATA. This paper shows that destruction, the reverse process of self-reproduction, is possible in Codd's space. In other words, a universal computer can self-reproduce and, hereafter, erase its offspring again. Self-reproduction is useful for highly computational programs while destruction permits adaptation to programs requiring a lot of storage space.

The first step in destroying a machine is to paralyse its activity. Then the destroying machine makes the passive network part of its own construction (destruction) arm, and retracts it.

CONTENTS

1. Introduction and Summary

2. Description and Paralysation of the Extended UCC
 - 2.1 New Components
 - 2.2 The Modifications of the UCC

3. Destruction
 - 3.1 Assumptions
 - 3.2 Cutting up a Straight Path
 - 3.3 Removing Path Components
 - 3.4 Justification of Transition Function Changes

1. Introduction and Summary

Parallel operation is one of the most characteristic features of cellular spaces. Universal Turing machines can be implanted into cellular spaces to make them computation-universal. However, Turing machines are sequential so that the power of parallel operation takes effect only on a level where several machines operate simultaneously. Therefore self-reproduction becomes a useful property. Assume a problem requiring a lot of computational work is to be computed by a cellular machine. It could first reproduce itself, say n times, and then the $n+1$ machines might start computing in a parallel manner.

Several computation-universal self-reproducing cellular machines have been designed (J. von Neumann [4] and J.W. Thatcher [3], E.F. Codd [1], A.R. Smith III [2]). The background material for this paper will be Codd's CELLULAR AUTOMATA [1]. The space described there is minimal in the sense that it is a strongly rotation-symmetric 8-5 space (each cell has 8 states and 5 neighbors) while von Neumann's and Smith's spaces are weakly rotation-symmetric 29-5 and not rotation-symmetric 7-7 spaces respectively. Compared with the latter Codd's space is not minimal if one applies the following complexity measures: (1) The ratio cellular time steps per simulated Turing machine time step ("distance to real-time simulation"), and (2), the number of cells required to simulate a Turing machine.

Even problem-oriented Turing programs are not very efficient. To use universal Turing machines for practical purposes is absurd. As

Smith points out, his 7-7 space is only capable of simulating a universal Turing machine, i.e. his cell constructions are highly Turing-machine-independent. For example, a (relatively small) 1000-state Turing program would generate a 1001-state cellular space. Moreover a real-time simulation of a universal Turing machine in the 7-7 space is, of course, not real-time with respect to the original Turing machine simulated by the universal machine.

Codd's self-reproducing universal computer-constructor (UCC) embedded in an 8-5 space is Turing-machine-independent, i.e. capable of simulating every Turing machine. It shows interesting structural similarities to real computers. The UCC offsprings are separated in such a way that they can operate without interference if desired, while the UCC copies in Smith's space have to share one common tape.

Though a reasonable technical realization has not at all been found, we think that among the cellular computers mentioned above Codd's UCC is best adapted to practical purposes. There is still a gap pointed out by Codd in one of the open questions concluding his work: "Can the partial transition function specified in Chapter 4 be extended (without adding new states to the space) so as to make possible the destruction of a machine which is active but not actively defending itself?" In other words, is there a reset-program that restores the initial situation of only one machine being in the space?

will only deal with a finite subspace, a

problem involving a lot of computation time has been computed in parallel by n reproductions of an original UCC, a second problem might need a great deal of storage space. Now one of the computers might destroy some of the copies again and use the space occupied by them previously as storage space.

Another application might be found in taking universal computing machines as simulators of biological cells or organisms. Destructing a machine and reconstructing it somewhere else may be regarded as a movement of the organism, moreover, one machine destroying another one might simulate the killing or dying of cells or organisms.

The present paper shows that the UCC in Codd's 8-5 space can be modified and extended in such a way that a UCC is capable of destroying an active copy of itself. This takes place in two main steps.

1. Paralysis: The destroying machine turns off the periodic emitter, the source for all signals, of the machine to be destroyed thus converting it to a passive network of paths, corners, T-junctions, and gates.
2. Erasure: A subroutine S enables the destroying machine to "cut up" a path. With the help of this subroutine the network can be removed.

From a different point of view, step 2 constitutes a way to erase an important subclass of the configurations over $\{0,1,2,3\}$. Before, only techniques for erasing $(0,1)$ configurations were known in Codd's cellular space. Comparing this result with von Neumann's 29-5 space

Let us note that today no way is known to me that shows how in von Neumann's space a (2-dimensional) passive (0,1) configuration, not to mention an active UCC, could be erased.

Let us fix the result of this paper precisely. The reader is assumed to be familiar with the definitions and terms introduced by Codd in [1]. We add the definition: The configuration c destroys the configuration d disjoint from c if there exists a time t such that

$$F^t(c \cup d) \upharpoonright \text{sup } d$$

is quiescent. In other words, c erases the support of d within t time steps. Although there are trivial cases, erasing is non-trivial in the following

Theorem: The transition function in Codd's 8-state, 5-neighbor space can be extended and slightly modified in such a way that there exists a self-reproducing universal computer-constructor (UCC) with the following property: Let there be in the construction site of a universal computer-constructor UCC_1 a second one, say UCC_2 , - accessible for UCC_1 - executing any computation or construction task. Then UCC_1 is able to destroy UCC_2 .

The information stored in UCC_2 's memory section and additional UCC's in the space are not affected by the destruction process. (Applying the definition exactly, one had to say: If the initial configuration of the whole space containing UCC_1 , UCC_2 , tapes, additional UCC's etc. is c , then $(c - UCC_2)$ - the configuration "outside of UCC_2 " - destroys UCC_2 .)

2. Description and Paralysis of the Extended UCC

In this chapter we are going to describe an extension of Codd's UCC with additional properties useful in chapter 3. We will show that the new UCC can be paralysed by the injection of a single signal (analogous to the activating signal) with the result that it becomes a passive configuration over $\{0,1,2,3\}$. In other words, all signals circulating in the UCC are annihilated. An end-of-paralysation signal is sent to the destroying machine.

2.1 New Components

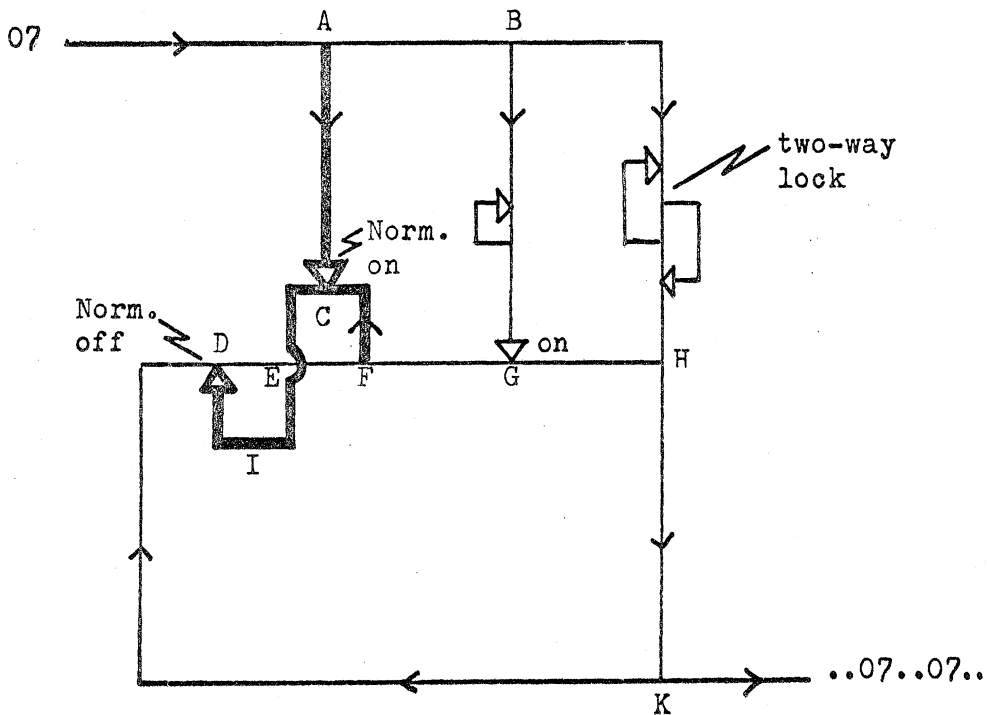


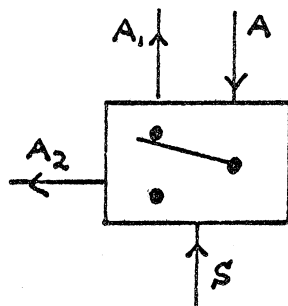
Fig. 1 Periodic emitter with turn off

Fig. 1 shows the periodic emitter with turn off. The first input signal 07 at A starts the periodic emission of 07 outputs at K until a second 07 input is given. Without the heavily printed lines AC and FCEID we have exactly the periodic emitter shown in [1], Fig. 5.7 .

Copies of the first 07 input turn gates C and G on and reach via junction H the points G and K. The one at G is annihilated, the other one begins to circulate clockwise. Every circulation produces (periodically) a copy leaving the circle at K and copies at F and H, which are annihilated by gate C and the two-way lock respectively.

Now the second input turns gate C off implying that gate D is turned on by the next copy produced at F by the circulating signal. Because we may assume that $FCEID < FGHKD$, the circulating signal is annihilated at D and the emitter stops working.

The component switch symbolized by



has two inputs A and S and two outputs A₁ and A₂ . An activating input 07 at S enables the switch to work properly: All signals (0s)_{s=4,5,6,7} entering the switch at A leave it unchanged at A₁ until a switch input 07 is given at S. The next input at A will be converted to 07 and leave

the switch at A₂. There is the restriction that the first input at A is a 07. This requirement, however, will be met automatically when using the component in the UCC.

Fig. 2 shows the details of the switch. From now on let us make the assumption, although not always necessary, that \longrightarrow represents a one-way lock.

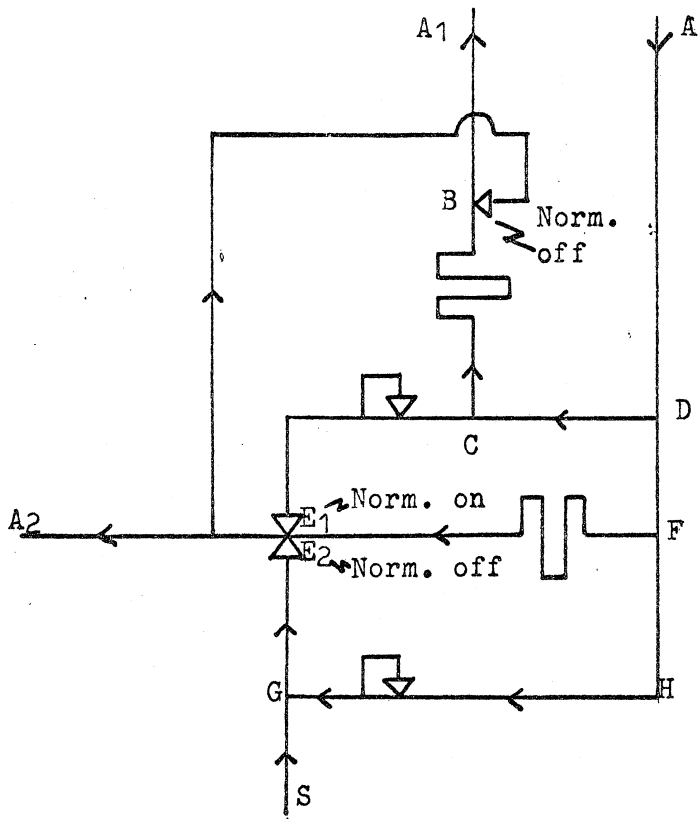


Fig. 2 Switch

The activating input at S turns gate E₂ on. The first input at A is a 07 signal duplicating at D, F and C. Copy 1 proceeds via H to gate E₂

turning it off. Copy 2 is annihilated by gate E_1 which was turned on before by copy 3 via C. This is guaranteed by the delay after F implying $DCE_1 \ll DFE_1$ and $DHGE_2 \ll DFE_2$. Finally, copy 4 leaves the switch at output A_1 . So do all the following inputs at A. Their remaining three copies, however, are annihilated between H and G, C and E_1 and at E_1 .

The situation changes when a switch input at S turns gate E_2 on thus converting the two gates E_1 and E_2 to a signal transformer type 7 ! The next input 0s at A is converted to 07 leaving A_2 and turning B on before the copy 0s from C leaves the delay after C ($DFE_1B \ll DCB$). Thus no output occurs at A_1 .

2.2 The Modifications of the UCC

We extend the memory section shown in [1], Fig. 6.2 by three additional construction paths C_1 , C_2 , and C_3 (though their purpose is destructive rather than constructive), a timing path T which is merely a straight path and 8 position tapes PX , $X \in \{C_1, C_2, C, C_3, D, K, M, P\}$ as shown in Fig. 3. The position tape PX , $X \in \{D, K, M, P\}$ contains the natural number n if the n-th cell of tape X is scanned by its tape path. The easiest way to represent n is a string of n '1's. Whenever tape path X moves one step to the right or to the left, the tape path PX does this too while at the same time, however, marking or erasing a '1' after or before the move respectively.

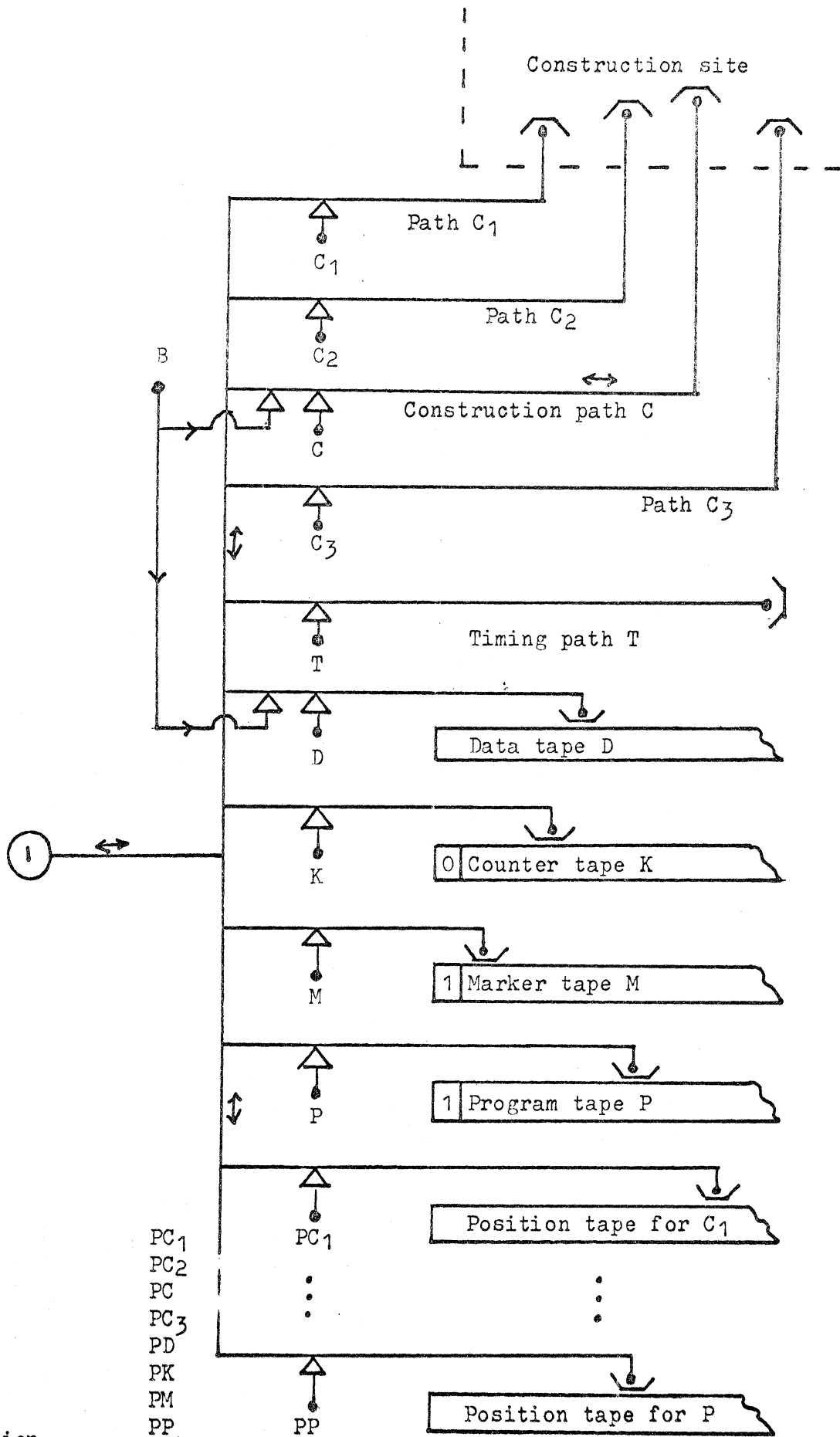


Fig. 3
Memory section

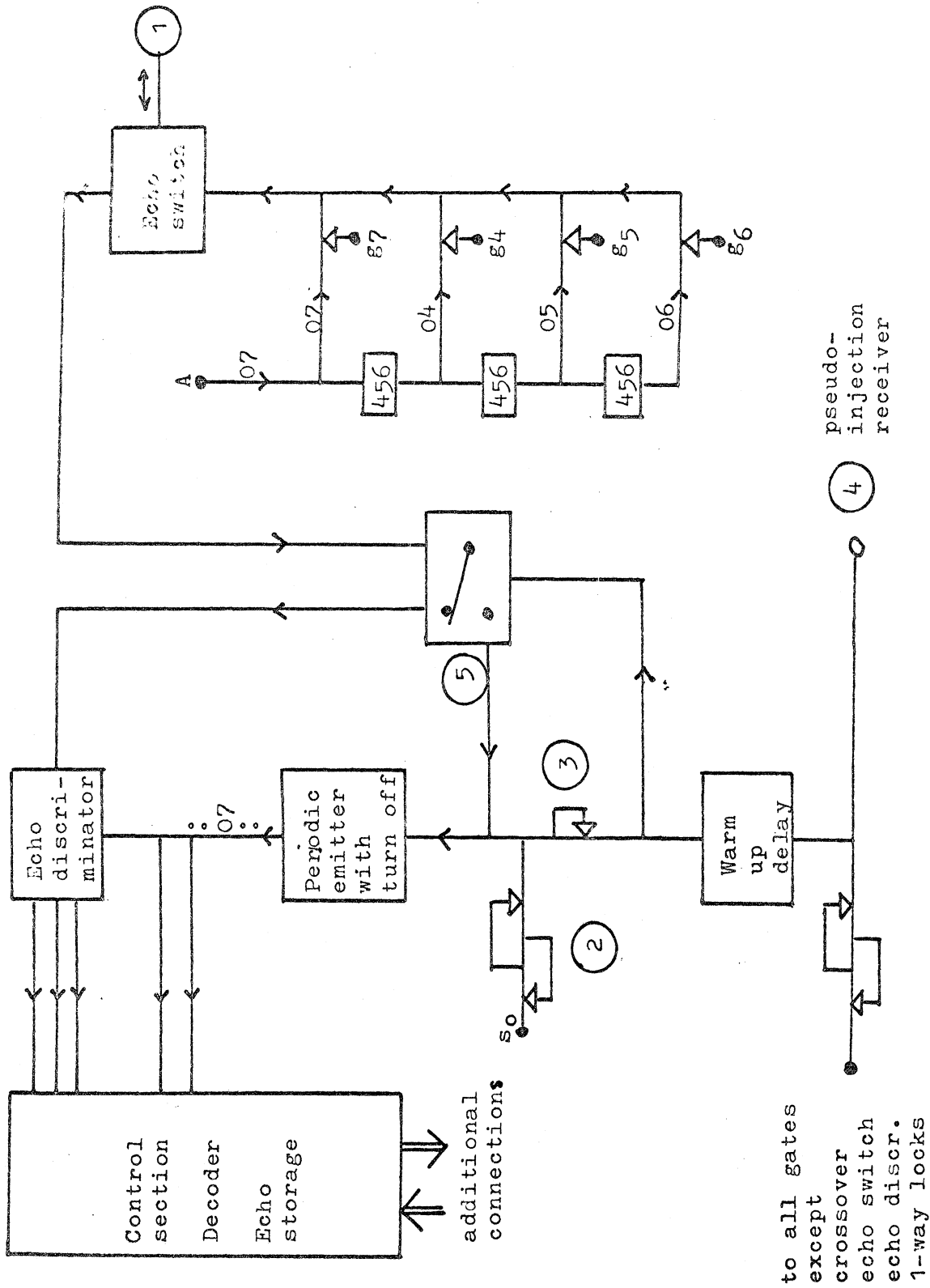


Fig. 4 Partial execution section

to all gates
except
crossover
echo switch
echo discr.
1-way locks

Analogous the position tapes PX, X = C₁, C₂, C, C₃ keep track of the positions of the construction paths. One possible way to do this is to represent 'extend' by '1', 'extend right' by '01', and 'extend left' by '001'. Whenever a retract operation is executed, the corresponding code '1', '01', or '001' has to be erased. For example, the position obtained by the instruction sequence x-x-x-xr-x-xl-x-x-r is represented by 1110110011 .

Let us now consider the differences between the execution section of the old UCC as designed in [1], Fig. 6.2. and the one of the new UCC in Fig. 4. The periodic emitter is replaced by one with turn off, we added a switch and the locks (2) and (3) and we replaced the linkage between the periodic emitter and the transformers type 456 as well as the subordinate-restored gate A by the simple input A. That avoids crossing a path with periodic signals proceeding on it. Furthermore we do not need an injection receiver because section 3 describes a way to connect and disconnect two paths.

How is the new UCC operating? Let there be in the construction site of UCC₁ the (0,1) skeleton of UCC₂. We show how UCC₁ activates UCC₂ and later on destroys it.

The pseudo-injection receiver (4) of UCC₂, an ordinary unsheathed path end, is connected with the construction path C of the stimulating machine UCC₁. The sheath signal 06 and the activating signal 07 are injected. The activating signal turns on the new periodic emitter and all gates as described in Fig. 4. Furthermore it calls for the first

microprogram step and activates the switch. Thus the switch now connects the memory section with the echo discriminator. Note that the first echo signal to pass back from the memory section is 07. After the injection of 06 and 07 the paths C_1 and C_2 of UCC_1 will disconnect its path C and the pseudo-injection receiver (4) of UCC_2 (see section 3.2) .

Program execution takes place almost like in the old machine. If a 07 signal is to be sent to the transformers 456, we do this directly rather than releasing a signal from the periodic emitter by means of a subordinate-restored gate. Moreover, the microprogram in the control section keeps the position tapes up to date whenever one of the construction paths or paths D, K, M, or P are moved. Note that the position tape PC_3 holds implicitly the information about the current length of the timing path T because we want to assume the following rule: The length of T is kept t units longer than the length of C_3 where t is a fixed positive integer.

Now the destructing machine UCC_1 connects its path C with the pseudo-injection receiver (4) again. We assume that the paralysation signal (a single 07) is not injected too early, i.e. UCC_2 is at least completely activated, thus either testing the start cell or executing a program or having finished the execution again testing the start cell.

The paralysation signal flips the switch. After an unknown but finite time an echo from the memory section enters the switch and leaves it via (5) converted to 07. This signal turns the periodic emitter off.

The last signals produced by the emitter are annihilated somewhere because the control section is waiting for a (never arriving) response from the echo discriminator.

Hence it is clear that no signal circulates in UCC_2 when the end-of-paralysation signal - a copy of the O7 output at (5) - leaves (4) towards UCC_1 . Paralysation has taken place.

At first sight, one might ask if one could not do without the switch. The paralysation signal could turn off the emitter directly. This would stop any activity except for a sense-and-wait signal which might already be on the way at the time the emitter is turned off. So UCC_1 had only to wait until the echo would return and die somewhere in the control section. The crux lies in the word "until". Since the length of a tape is unbounded, only the returning signal itself can tell after what (finite) time a sense-and-wait operation is terminated.

3. Destruction

In this chapter we show how the paralysed UCC_2 , a network of sheathed paths, can be destroyed. This will complete the proof of our theorem. Note that any arbitrarily fancy signal sequence which might be able to dissolve a path when passing, would fail at the first of many one-way locks. Therefore the network can be destroyed only "from the outside".

3.1 Assumptions

We make the following assumptions about the extended UCC.

1. The distance between two corners or a corner and a gate node is at least 23 units. (A corner may have two or three arms. In the latter case it is usually called T-junction.)
2. All paths running parallel have a distance of at least 23 units.
3. Turning on a gate does not happen while a signal is passing on the subordinate path. Thus we avoid the neighborhood state 2 0232 . This requirement is met if we design the particulars of the UCC carefully and provide a sufficiently long distance between two successive signals.

The only exceptions to assumptions 1 and 2, which provide enough operation space for the destruction paths, concern the construction and tape paths. However, also they should avoid zigzagging (produced e.g. by 'extend right'-'extend left') and should not have their sheath in contact with that of any other path.

In passing we note that the only uncertainty about state and location of a paralysed UCC concerns the positions of the memory paths and the states of the gates.

Under these assumptions we can now reduce the destruction of a paralysed UCC to the problem how to "cut up" a straight path. The solution of this problem follows.

3.2 Cutting up a Straight Path

We are going to extend and slightly modify the partial transition function f given in [1], table 4.10 which enables us to define 10 subroutines useful in cutting up a straight path.

Subroutine S1 - extend to a path: Fig. 5 shows the extension of a construction arm to a path sheath using the signal sequence $x = (07-06)$. Though the old UCC defined in [1] does not require this operation, no extension of f is necessary.

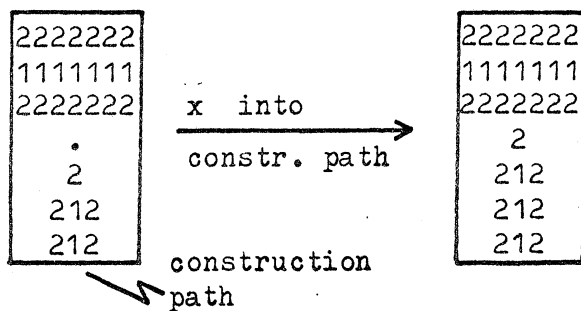


Fig. 5 Extend to a path

Subroutine S2 - retract from a path: This operation is inverse to S1. The required signal sequence is $r = (04-05-06-06)$. However, the transition function f has to be extended.

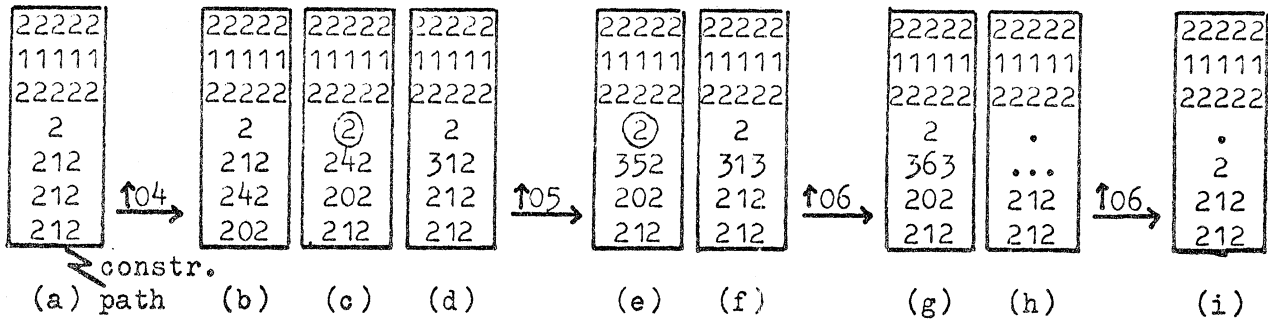


Fig. 6 Retract from a path

Fig. 6 gives detailed shots of the retract operation. $\uparrow 0s$ means that we send signal 0s in the construction path. The additional transitions are

$$\begin{array}{l} 2 \ 0204 \ 2 \quad c \ (4/3) \\ 2 \ 0205 \ 2 \quad e \ (4/3) \end{array}$$

For example, in Fig. 6.e the cell in row 4 and column 3 emphasized by $\textcircled{}$ faces the undefined neighborhood state 2 0205 .

Subroutine S3 - cut away first '2' of a sheath: The initial situation (Fig. 7) is produced by subroutine S1 and requires a turned on gate in the construction path.

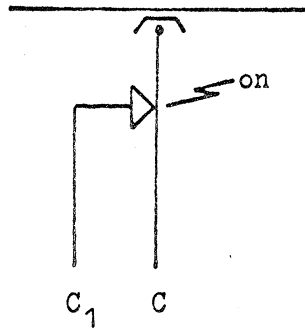


Fig. 7 Initial situation for S₃, S₄, and S₅

As Fig. 8 shows, the signal sequence $c_2 = (06-04-04)$ (cut away '2' of a sheath) produces an undesired echo. Gate C₁, however, will annihilate this echo immediately. Afterwards the gate may be turned off and path C₁ can be retracted using subroutine S₂.

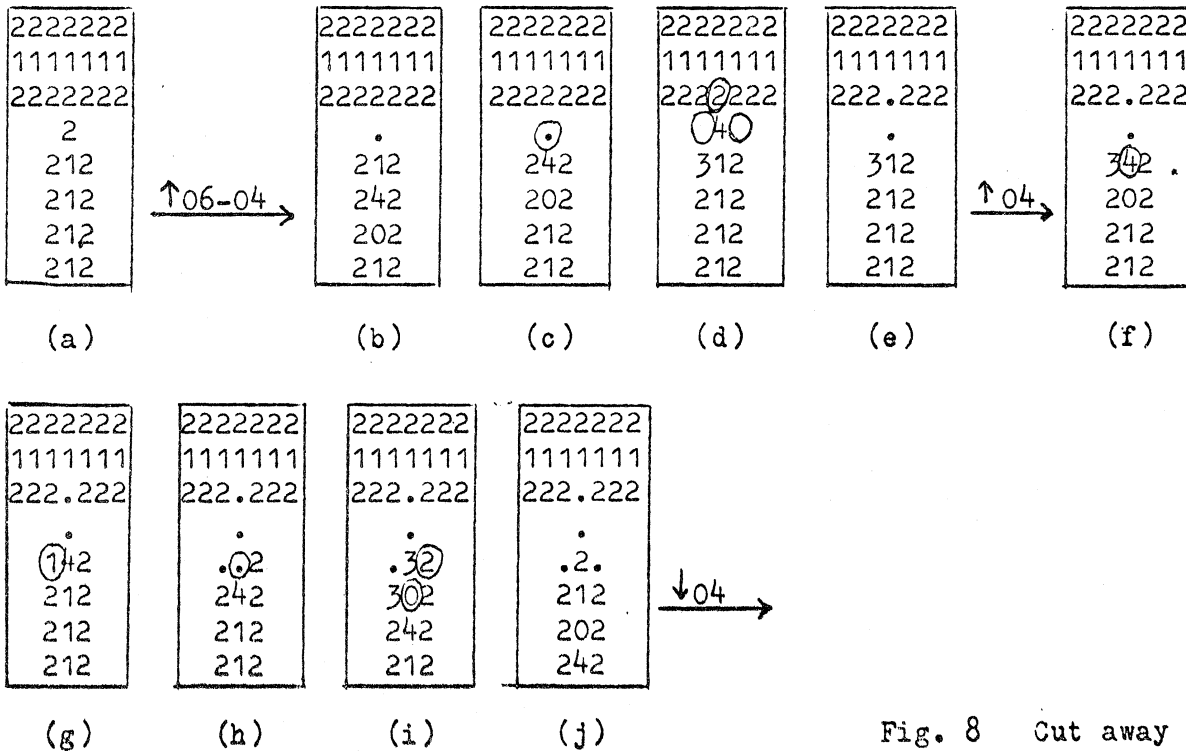


Fig. 8 Cut away first '2' of a sheath

The required new transitions are

0	0204	4	c	(4/4)
≠ 2	1242	0	d	(3/4)
0	0243	0	d	(4/3)
0	0242	0	d	(4/5)
4	0203	4	f	(5/4)
1	0042	0	g	(5/3)
0	0024	3	h	(5/4)
≠ 2	0023	0	i	(5/5)
0	2433	1	i	(6/4)

The two transitions denoted by ≠ are a modification rather than an extension of f. We will take special care of them in section 3.4 .

Subroutine S4 - cut away additional '2' of a sheath on the right of a gap: Using the same initial situation (Fig. 7) and the same signal sequence o2 as in the last subroutine, we add the two transitions

2	0124	0	d	(3/4)
0	0043	0	d	(4/3)

to obtain the results shown in Fig. 9 .

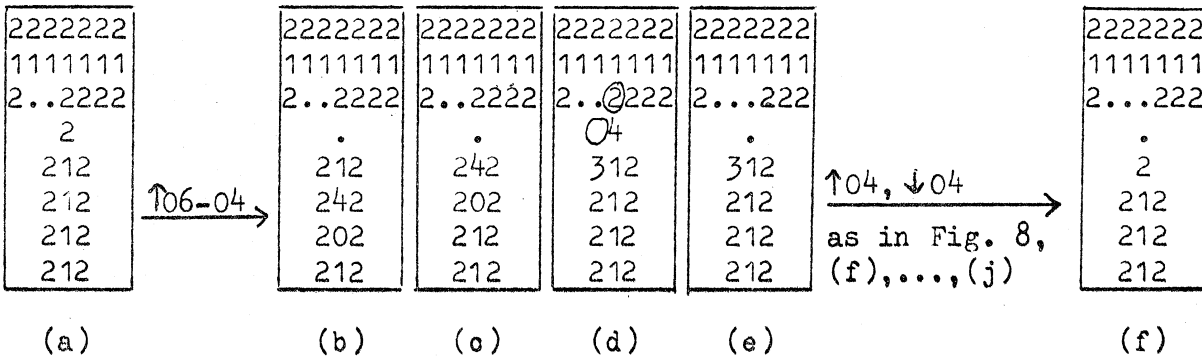


Fig. 9 Cut away additional '2' of a sheath on the right of a gap

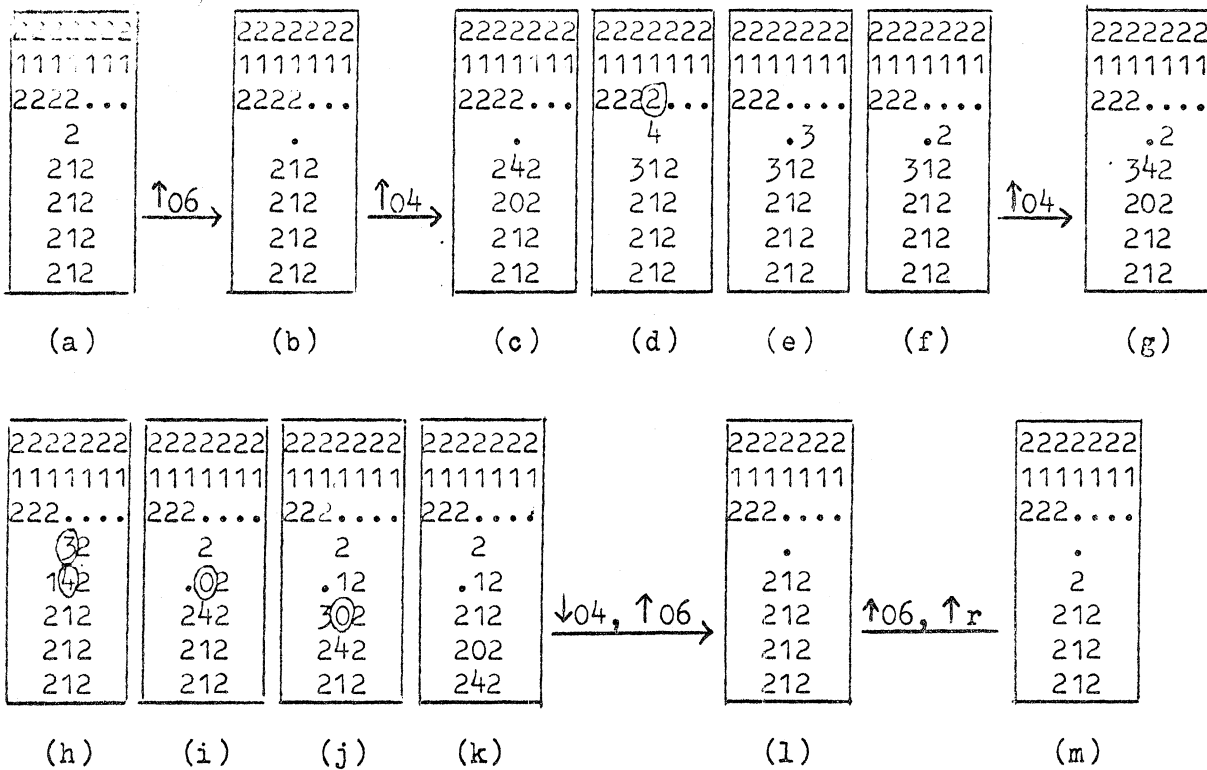


Fig. 10 Cut away additional '2' of a sheath on the left of a gap

Subroutine S5 - cut away additional '2' of a sheath on the left of a gap: Again, we start in the initial situation of subroutine S3, but we use an extension of c2, namely the signal sequence c21 = (06-04-04-06-06), to induce the construction path to execute the desired cut away operation (Fig. 10). For a proper operation of the subroutine we have to add the transitions

- 2 0421 0 d (3/4)
- 3 0024 2 h (4/4)
- 4 1132 0 h (5/4)
- 0 0224 1 i (5/4)
- 0 1243 1 j (6/4)

Furthermore we should take care that the second 04 of c21 is not too closely followed by 06 so that a collision between the latter and the 04 echo is avoided.

Subroutine S6 - cut away first '1' of a path: Assume sufficiently many 2's of the path sheath have been cut away. Extending the construction path to the '1' shown in Fig. 11.a is the same as if a (0,1) configuration is approached. Moreover, we can use the erase sequence $e = (06-07-04-05-06-06)$. However, the transition table requires two additional rows:

1 0124 1 h (2/4)
 1 0421 1 h (2/6)

Fig. 11 gives the details.

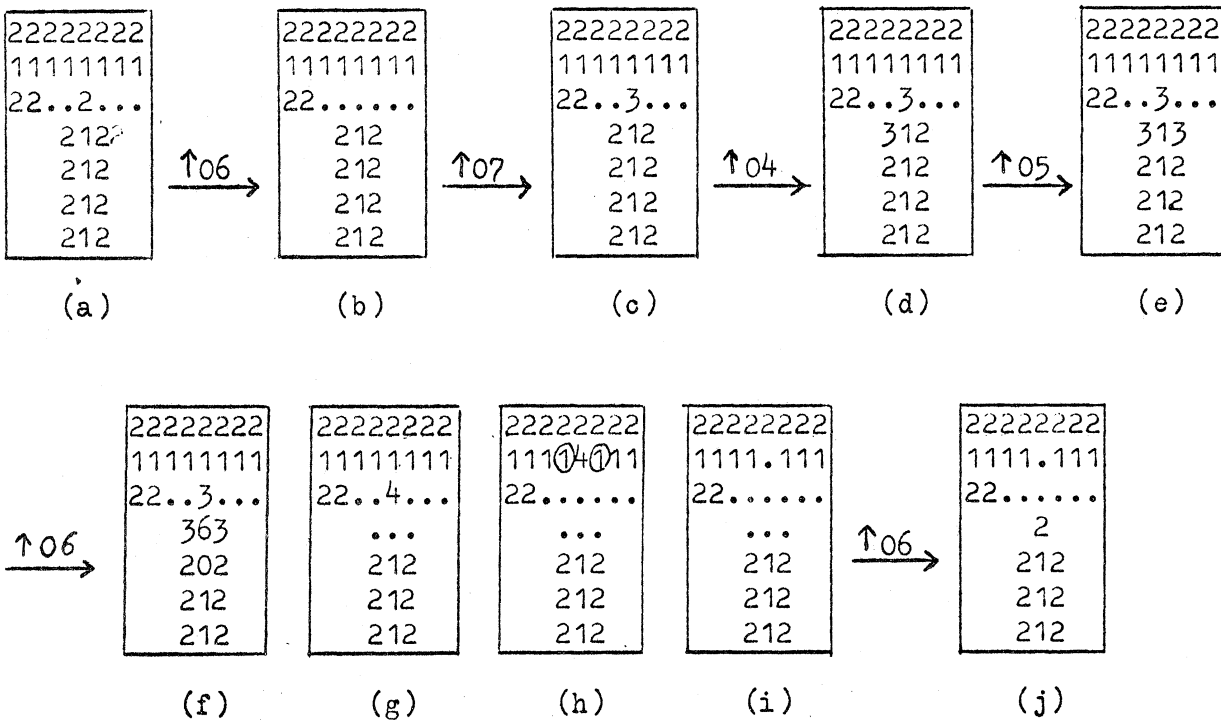


Fig. 11 Cut away first '1' of a path

Subroutine S7 - cut away additional '1' of a path on the right of a gap: The same initial situation as in the previous subroutine, the same signal sequence e and only one more transition, namely

1 0214 0 b (2/4)

are required to perform subroutine S7 as shown in Fig. 12.

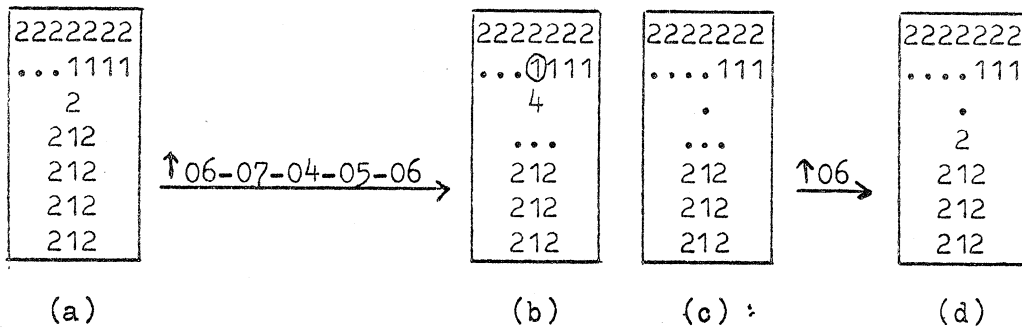


Fig. 12 Cut away additional '1' of a path on the right of a gap

Subroutines S6 and S7 enable the construction arm to cut away the core of a path from the left to the right. Analogously, S3 and S4 are sufficient to remove the sheath of a path. One might ask, why S5 is necessary at all. However, the main subroutine S will call S5 when generating norm ends.

Subroutine S8 - cut away first '2' of a 2-chain: After removing the sheath and the core of a path a simple chain of 2's is left over. Subroutine S8 cuts away the first '2' of such a chain, namely the one determined by the initial position of the construction path in Fig. 13.

Subroutine S1 is useful in obtaining this position.

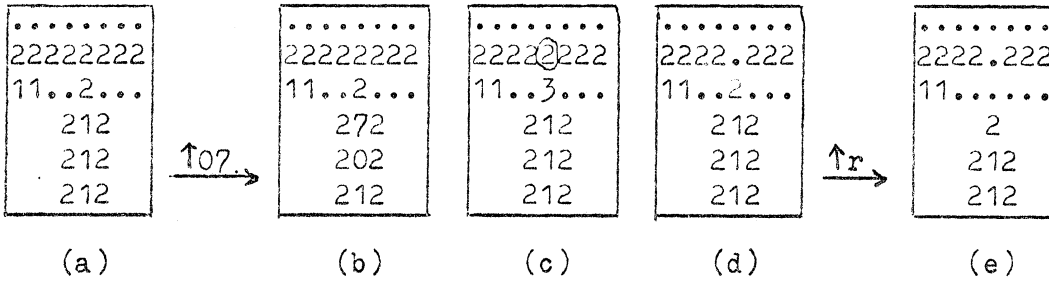


Fig. 13 Cut away first '2' of a 2-chain

We call the required signal sequence c2c (cut away '2' of a chain) which is 07 concatenated with the retract sequence:
 $c2c = (07)-r = (07-04-05-06-06)$. This time we have to alter a row of the transition table:

$$\neq 2 \ 0232 \ 0 \quad c \ (2/5)$$

Subroutine S9 - cut away additional '2' of a 2-chain: The sequence c2c also permits the removal of a '2' of a 2-chain located to the right or to the left of a gap, as is demonstrated in Fig. 14 and Fig. 15 respectively. However, the subroutine is not able to annihilate a single '2'.

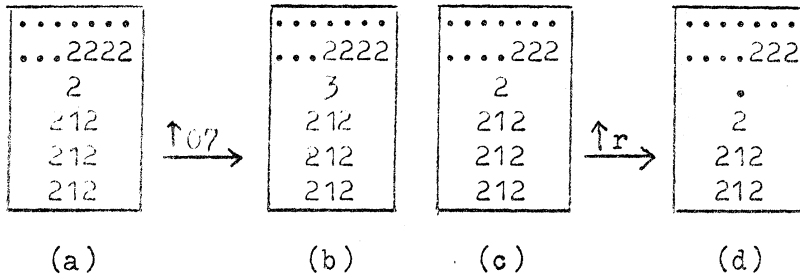


Fig. 14 Cut away additional '2' of a 2-chain on the right of a gap

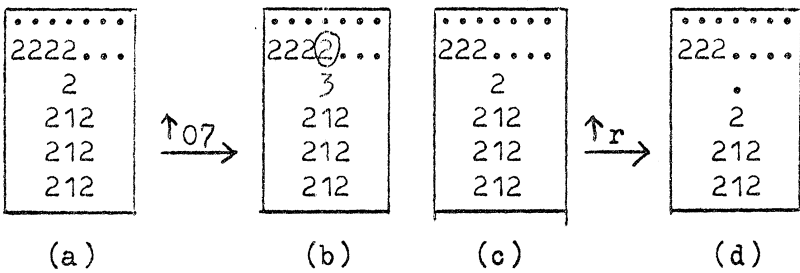


Fig. 15 Cut away additional '2' of a 2-chain on the left of a gap

In Fig. 15.b there occurs a neighborhood state whose image under the transition function is now defined differently than in [1] :

$$\neq 2 \ 0032 \ 0 \quad b \ (2/4)$$

Subroutine S - cut up a straight path and generate norm ends: This subroutine calling all the previous ones is the heart of the process of destruction. Fig. 16 shows how S cuts up a straight path. Not explicitly mentioned are the various extend and retract operations executed by the destructing paths. For some of them S1 and S2 are called. Paths C1 and C are omitted in Fig. 16.

22222222222222222222222222222222
11111111111111111111111111111111
22222222222222222222222222222222



22222222222222222222222222222222
11111111111111111111111111111111
2222.2222222222222222222222222222



22222222222222222222222222222222
11111111111111111111111111111111
2222.....2222



22222222222222222222222222222222
111111.11111111111111111111111111
2222.....2222



22222222222222222222222222222222
111111.....111111
2222.....2222



22222222.222222222222222222222222
111111.....111111
2222.....2222



22222222.....22222222222222222222
111111.....111111
2222.....2222

Fig. 16 Cutting up a straight path



Now the construction path C can easily move through the gap and produce norm ends defined by Fig. 17 .

```

2222.....2222
111111.....111111
2222.....2222

```

Fig. 17 Path gap with norm ends

Norm ends should be distinguished from standard ends symbolized by  and  respectively. Referring to the letters in Fig. 18, path C may cut away the 2's under letter

a , b , c , d , e , f , g , h

calling subroutine

S9, S9, S4, S4, S9, S9, S5, S5

respectively, in the indicated order.

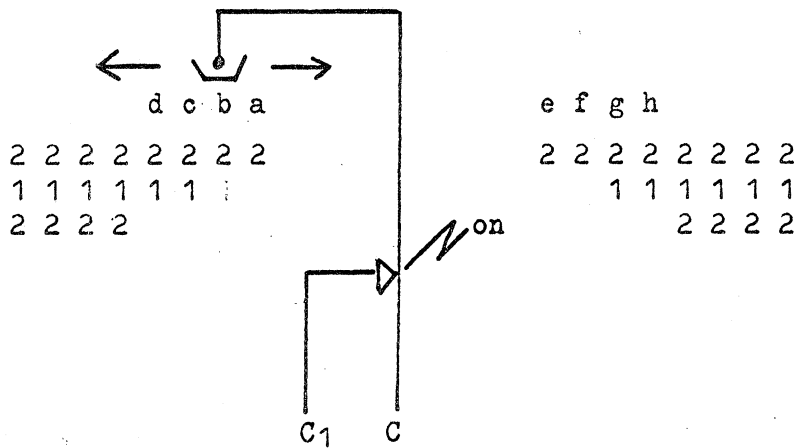


Fig. 18 Generating norm ends

Note that this is the only time where S5 and the second application of S9 are needed.

It is no problem to approach a norm end and to connect it with the corresponding path using the signal sequence $x = (07-06)$ as indicated in Fig. 19. (This happens to be the same sequence as is required for the operation 'extend'.)

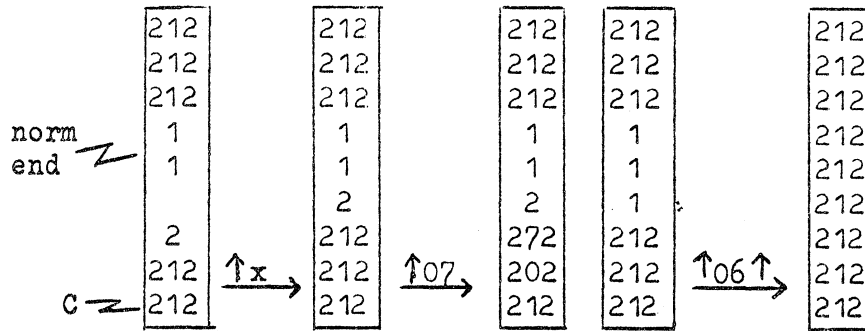


Fig. 19 Connecting with a norm end

The sheathing signal 06 proceeds to the top until it is annihilated by a gate or used for other purposes.

3.3 Removing Path Components

Taking advantage of the assumptions made in 3.1, we can always find enough space to apply subroutine S. Because the destroying machine UCC_1 knows the parameters of the network of UCC_2 , it can remove this network

from the outside inwards step by step.

1. Removing a corner.

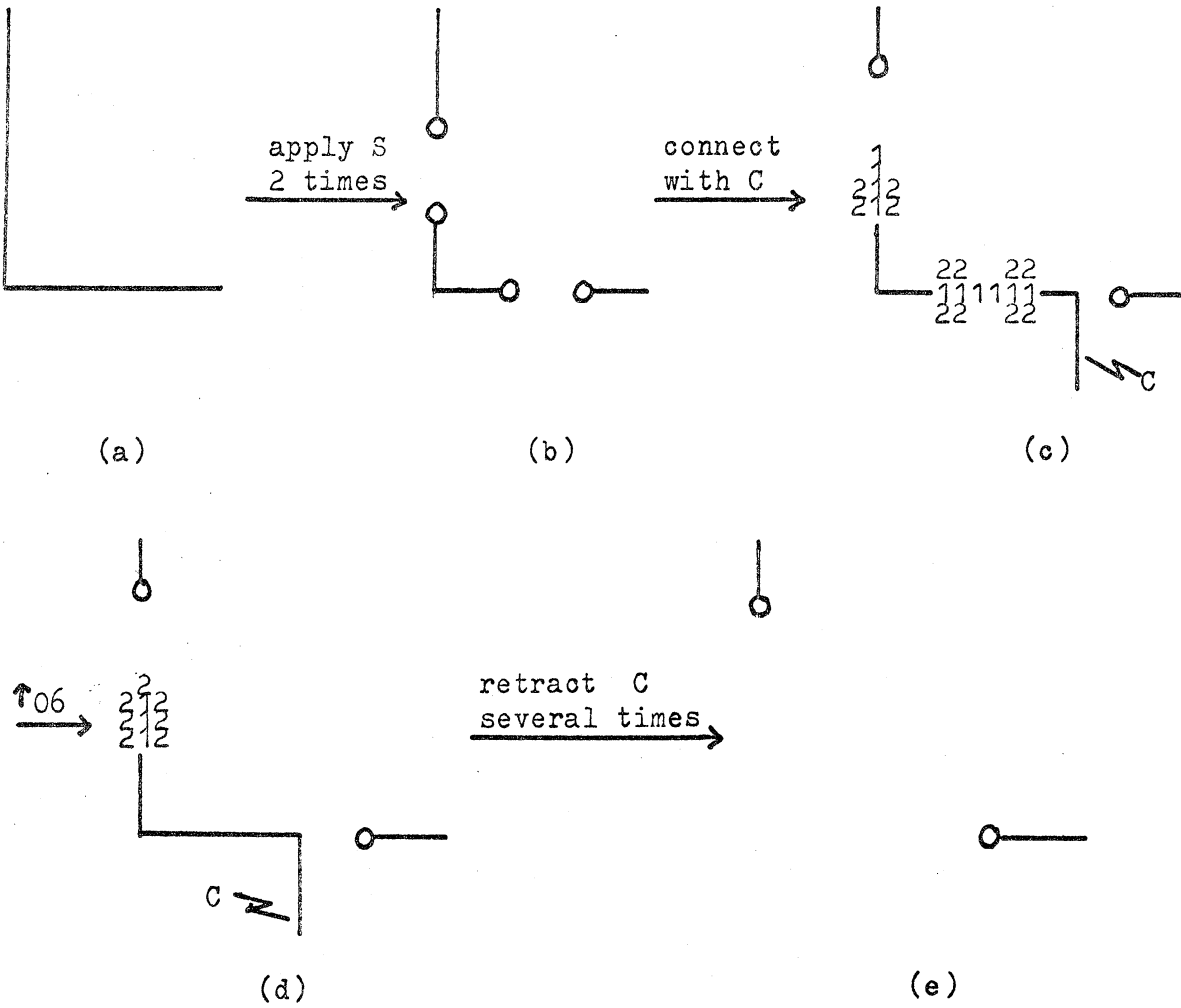


Fig. 20. Removing a Corner

Fig. 20 demonstrates how a corner may be annihilated. The two arms of the corner are cut up and path C connects itself with one of these arms. Now the corner is a part of path C

(Fig. 20.d) . It can be retracted easily.

2. Removing a T-junction. Let us consider Fig. 21 .

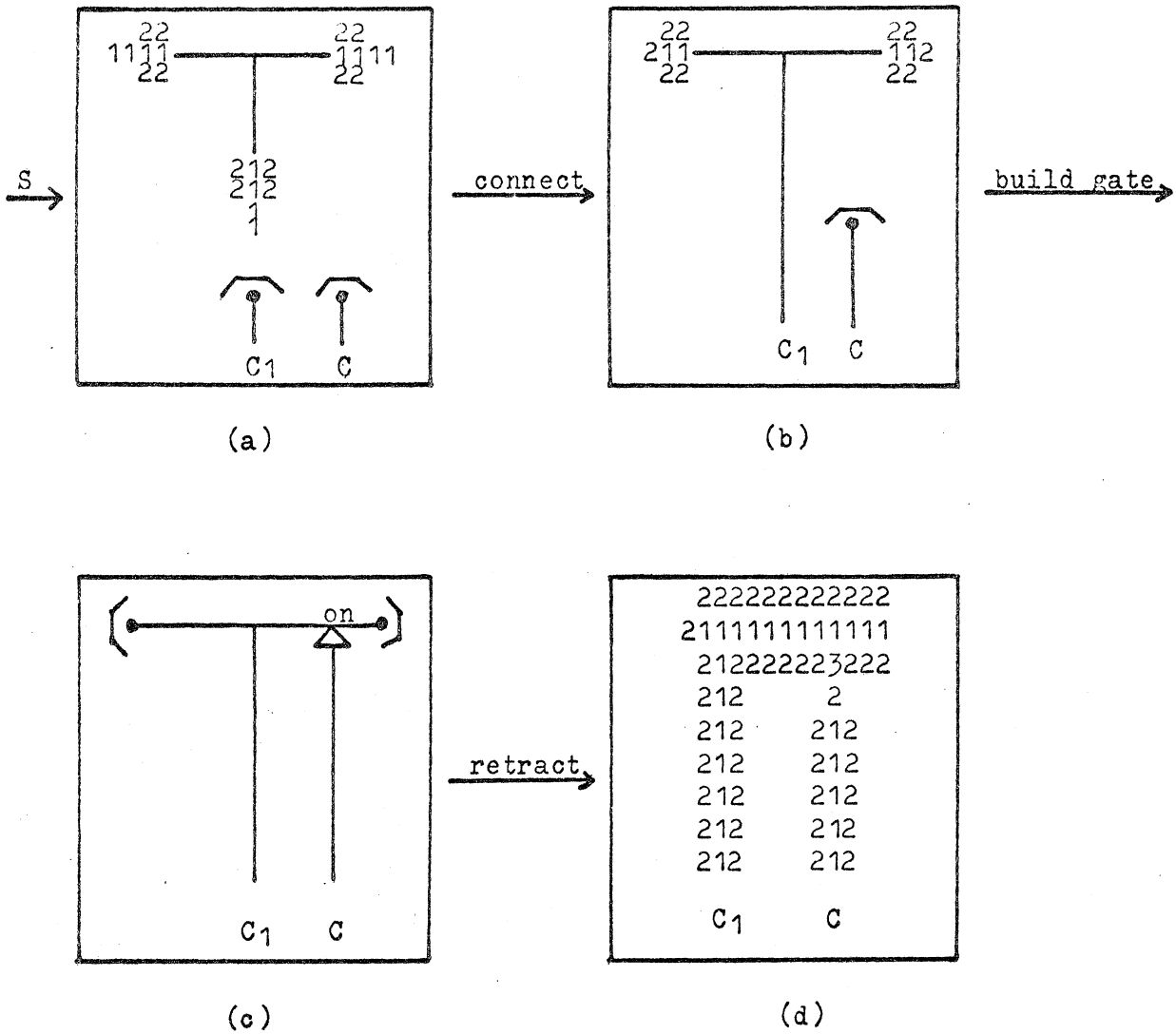


Fig. 21 Preparations for removing a T-junction

In (a) subroutine S has cut up the three arms of the T-junction. C₁ connects itself with one of them and converts the norm ends of the remaining two arms into standard ends (b). Approaching the right T-arm path C generates a gate and turns it on (c). Thus any signal sequence sent into path C₁ affects only the left arm of the T-junction. We can forget about the copies proceeding to the right. In (d) the left arm has been retracted as far as possible sending retract sequences into path C₁. Fig. 22 shows how the signal sequence rt = (05-06-06) (retract T-junction) retracts the remaining stump.

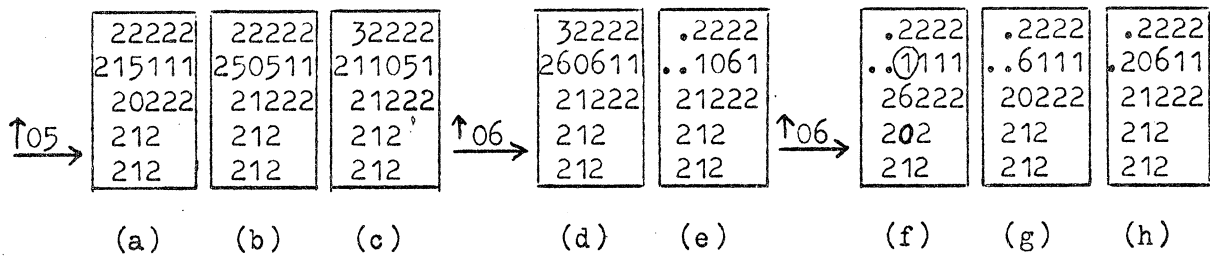


Fig. 22 Retract a T-junction

The result is a corner connected with (\longleftrightarrow being a part of) path C₁. Path C turns the gate off and retracts so that C₁ is able to remove the corner. We need only one additional transition:

1 0216.6 f (2/3)

3. Removing a gate.

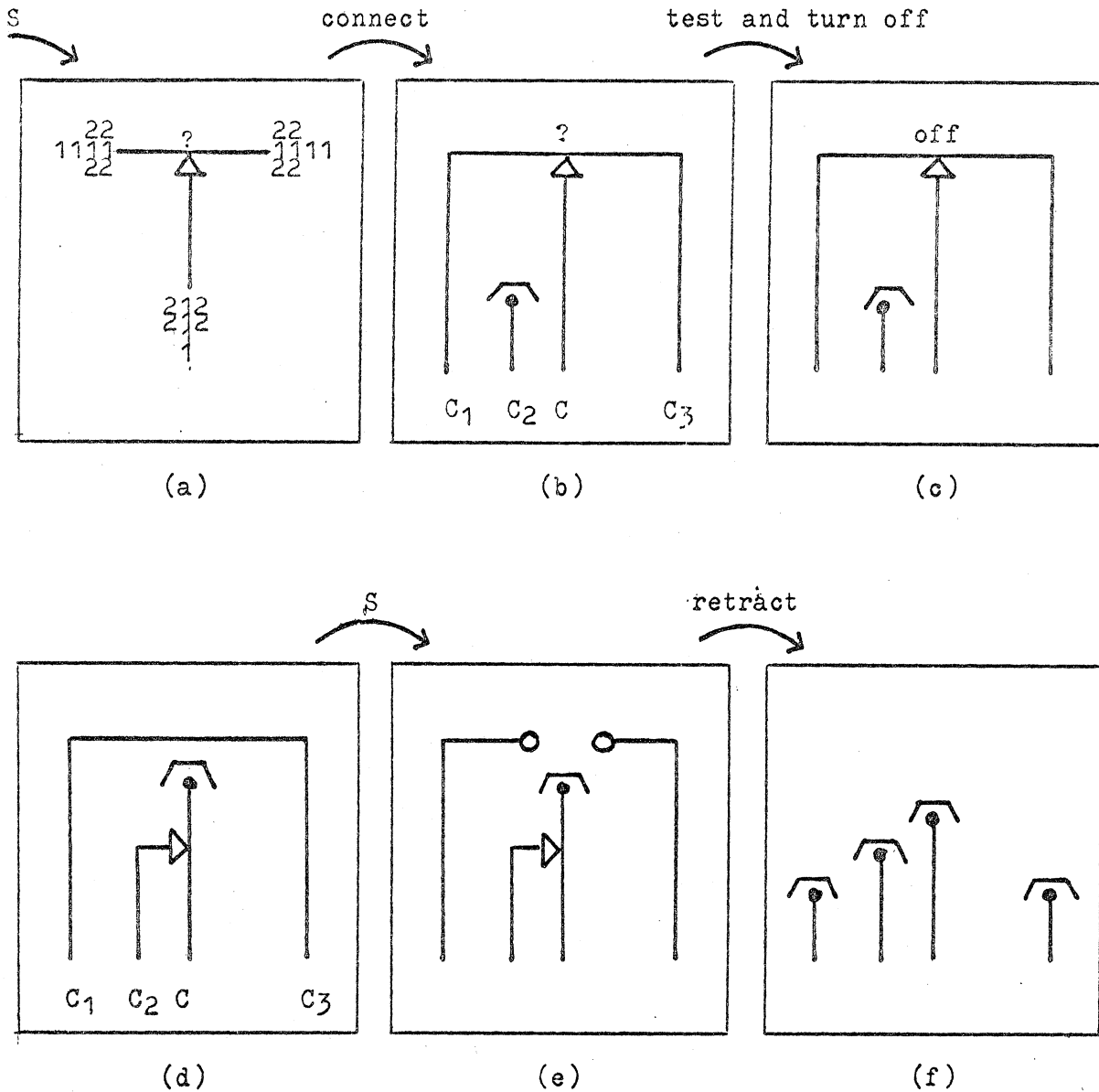


Fig. 23 Removing a gate in unknown state

Unfortunately, the destroying UCC₁ does not know the state of a gate to be removed because we want to enable UCC₁ to start the destruction process at any time, regardless which program or which step within

this program UCC_2 is executing. This is the only case where all four construction paths and the timing path are required. Fig. 23 shows the main steps.

Subroutine S produces (a) . After C_1 , C, and C_3 are connected as shown in (b) , a test signal may be sent into C_1 . If the gate is off, it will return, otherwise the gate will annihilate it. However, we do not know how long to wait for a (perhaps never) returning test signal.

Now the timing path T turns out to be very useful. It is not difficult to keep the length of C_1 less than the length of C_3 whenever a gate is to be removed. Thus

$$\text{length}(C_1) \leq \text{length}(C_3) = \text{length}(T) + t^+$$

holds for some fixed positive number t.

Assume the destroying machine UCC_1 injects first the sequence 'sense and wait' (sw) into T and then a single 07 into C_1 . Immediately afterwards UCC_1 turns off gate T in Fig. 3 controlling T. There are two possibilities:

Case 1: The tested gate is turned off. The test signal 07 returns from the memory section. Furthermore a copy of it enters path T because gate T is in the off-state, and collides somewhere with the echo 06 generated at the end of path T. Providing the transition

⁺ By convention, let $\text{length}(C_1)$ and $\text{length}(C_3)$ be constant while steps (b) through (f) in Fig. 23 are executed.

1 2627 7

both signals annihilate each other.

Case 2: The tested gate kills the test 07 because it is turned on. The echo 06 returns from the memory section thus indicating case 2.

In both cases a cap after sense sequence (cs) should restore the standard end of T. An analysis of the gate behavior yields the state diagram of a gate (Fig. 24).

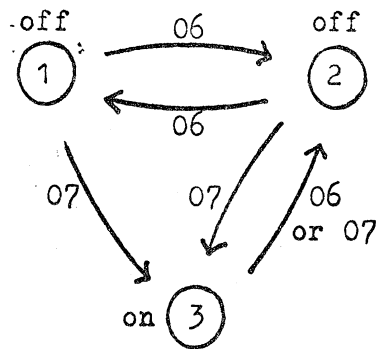
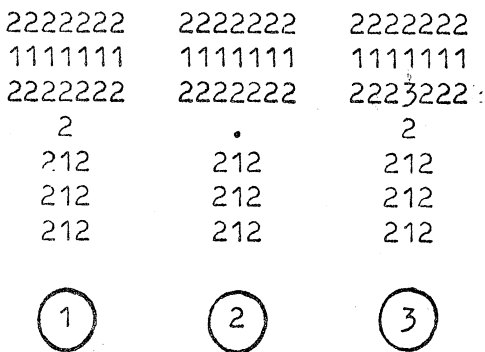


Fig. 24 State diagram of a gate

Applying the sequence $g_0 = (07-06-06)$ in case 1 (off) and $g_1 = (06-06)$ in the second case (on), we may be sure that the gate is in state ① (Fig. 23.c). Now it is easy to cut up the connection between C_1 and C_3 by use of C_2 and C (Fig. 23.e). Finally we obtain Fig. 23.f.

In this section and several times before we described the operation of the control section verbally. E.g. "The microprogram in the control section keeps the position tapes up to date ... " It should be clear that such cases require an extension of the control section, additional UCC commands, an enlarged decoder, and many little alterations. We do not go into these details because essentially they are treated in [1] .

4. Removing tape and construction paths. Whenever UCC₂ is paralysed it stops working after a sense and wait operation is executed. Therefore, UCC₁ should send a cap after sense sequence into the memory section of UCC₂ .

Step 1: Tape path PP_i of UCC₂ is disconnected from the memory section of UCC₂ and connected with path C₁ of UCC₁ . (The control of arm PP is transferred to UCC₁ .) C reads the number n on tape PP approaching PP from the bottom, and C₁ retracts n+1 times. (Note that path PP always scans the right neighbor cell of the n-th '1' of tape PP.)

Step 2: Applying step 1 analogous paths PM, PK, PD, PC₃, PC, PC₂, and PC₁ are removed.

Step 3: Using the information on the position tapes again, UCC₁ retracts P, M, K, D, T, C₃, C, C₂, and C₁ .

At this point we may recall the assumption of the theorem that UCC₂ - the machine to be destroyed - is executing any computation or construction task, that it is not, however, for its part destroying

a machine UCC₃ .

From a practical point of view this seems to be an unimportant restriction. However, let us assume for a moment that UCC₂ for its part is destroying a machine UCC₃ . In particular, UCC₂ might just have connected its path C₁² with the path PP³ of UCC₃ . At this moment, the information on the position tape PC₁² which determines the position of C₁² becomes completely false !

We will only briefly sketch the method by which we can extricate ourselves from this mess. Assume UCC₂ connects one of its construction paths X with a norm end of UCC₃ at some point y such that during the next sense and wait operation executed by UCC₂ the shape of X is undetermined by PX or X does not have a standard end. This case occurs if X is connected with a tape or construction path or participates in removing a gate. Then a break symbol is printed on PX. Now UCC₁ is able to remove that part of X which belongs to UCC₂: UCC₁ reads the break symbol which is the rightmost information on PX. Using the information about the shape of X, UCC₁ cuts up X at exactly the previous boundary position y and removes it.

5. Removing a signal transformer type 7 is no problem.

3.4 Justification of Transition Function Changes

At the beginning of this section we list two tables. Table 1 summarizes signal sequences necessary for destruction.

x	07-06	1. extend to a path 2. connect with a norm end
r	04-05-06-06	retract from a path
c2	06-04-04	cut away first '2' of a sheath or on the right of a gap
c2l	06-04-04-06-06	cut away '2' of a sheath on the left of a gap
e	06-07-04-05-06-06	cut away '1' of a path core
c2c	07-04-05-06-06	cut away '2' of a 2-chain
rt	05-06-06	retract T-junction
g0	07-06-06	turn off if off
g1	06-06	turn off if on

Table 1 Signal sequences necessary for destruction

Table 2 lists the additional transitions and those defined differently in [1].

2 0204 2	S2	3 0024 2	S5
2 0205 2	S2	4 1132 0	S5
0 0204 4	S3	0 0224 1	S5
≠ 2 1242 0	S3	0 1243 1	S5
0 0243 0	S3	1 0124 1	S6
0 0242 0	S3	1 0421 1	S6
4 0203 4	S3	1 0214 0	S7
1 0042 0	S3	≠ 2 0232 0	S8
0 0024 3	S3	≠ 2 0032 0	S9
≠ 2 0023 0	S3	1 0216 6	rt
0 2433 1	S3	1 2627 7	collision
2 0124 0	S4		
0 0043 0	S4		
2 0421 0	S5		

Table 2 Additional transitions and alterations

We have to justify the four alterations marked by '≠'. This leads to the question: Which of the transitions defined in [1] are really necessary? The following situations were simulated on a digital computer.

- a. Signals 04, 05, 06, and 07 proceeding straight ahead and passing a corner and a T-junction on every possible arm.
- b. Collisions of 06 with 60 and 07 with 70 on a straight path, at a corner, one and two units away from a corner, at a T-junction and one and two units away from a T-junction.
- c. 06 sheathing a network (including gates), analogous to a. and b.
- d. Sending 06 and 07 into all kinds of path ends.
- e. Sending 06 and 07 into the control path of a gate in state $\textcircled{1}$, $\textcircled{2}$, and $\textcircled{3}$ of Fig. 24.
- f. Os passing a turned on gate to the left and to the right and a signal transformer type 7 ($s = 4,5,6,7$).
- g. Executing x, xr, xl, r, rr, rl.
- h. Executing sw, cs, m, e (64 possibilities).

In marking every transition actually needed, it was found that many transitions in [1] were not used. This may be due to various reasons:

1. Configurations like paths running sheath adjacent to sheath, zig-zagging in a path (produced by xr-xl or xl-xr), a gate close to a corner, turning on a gate while a signal is passing on the subordinate path etc. make the construction of a UCC easier, but they are not necessary.
2. Systematically generated transitions using (0s)_{s=4,5,6,7} create redundancy. For example, collisions of 06's and 07's are, collisions of 04's and 05's are not required.
3. The short table 4.8 in [1] lists all exceptions from the rule: Neighborhoods over {0,1,2,3} remain unchanged. This defines all transitions over {0,1,2,3} and thus, of course, more than necessary.
4. An injection receiver was needed before.

In particular, the four alterations (\neq) in the transition table were not used. In a second computer simulation run the modified and extended transition table was loaded. Again, the situations (a),..., (h) were simulated, but also the subroutines S1,...,S9, and S and the removing of a path, a corner, a T-junction and a gate were tested. All operations were executed correctly.

Because this test included all neighborhood states occurring during the operation of a modified UCC as described in the previous chapters, the alterations of the transition table are justified.

BIBLIOGRAPHY

1. Codd, E.F., Cellular Automata, Academic Press, New York (1968).
2. Smith, A.R., III, "Simple Computation-Universal Cellular Spaces and Self-Reproduction", Proceedings of the IEEE Ninth Annual Switching and Automata Theory Symposium, pp. 269-277.
3. Thatcher, J.W., "Universality in the von Neumann Cellular Model", Technical Report 03105-30-T, ORA, The University of Michigan, 1964.
4. von Neumann, J., "Theory of Automata: Construction, Reproduction, Homogeneity", Part II of The Theory of Self-Reproducing Automata, edited by A.W. Burks, University of Illinois Press, Urbana, Illinois (1966).

UNIVERSITY OF MICHIGAN



3 9015 03127 2837