

CHURCH-TURING THESIS: THE TURING IMMORTALITY PROBLEM  
SOLVED WITH A DYNAMIC REGISTER MACHINE

**CHURCH-TURING THESIS COROLLARY 9.1**

Given any Turing machine  $(Q, A, \eta)$  as input, if there is a new computing machine using a finite number of computing elements and in a finite number of computing steps can determine whether  $(Q, A, \eta)$  has an immortal configuration, then the existence of this new computing machine explicitly disproves the Church-Turing thesis.

PROOF. This follows immediately from **8.44** and **8.45**

In this section, a *dynamic register machine* (DRM) is presented. Then in the following section a dynamic register machine program is demonstrated that can perform the following computation: for any Turing machine  $(Q, A, \eta)$  as input, the dynamic register machine program can execute method **8.36** in a finite number of computational steps. This explicit demonstration proves that the Church-Turing thesis is false.

The *dynamic register machine* is a new computing machine where its program can change while the machine is executing. It is an enhancement of the register machine conceived by Shepherdson & Sturgis. See [STURGIS].

The DRM has an unbounded number of *registers* labelled  $R_0, R_1, R_2, \dots$  each of which contains a natural number. There are seven *types* of instructions. A *finite sequence* of instructions is a *DRM program* denoted as P. The formal language for the dynamic register

machine is represented as S-expressions: register  $R_n$  as  $(R\ n)$ . The contents of all of the registers may be represented as a list of natural numbers, for example  $(17\ 3\ 22\ 5\ \dots)$ .

**INSTRUCTION 9.2** [0] *Constant Instruction*  $(C\ m\ v)$

For each pair of natural numbers  $m$  and  $v$ , the *constant instruction*  $(C\ m\ v)$  stores natural number  $v$  in register  $(R\ m)$  i.e. the contents of register  $R_m = v$ .

For example, if the contents of the registers are  $(17\ 2\ 671\ 3\ 81\ 95\ \dots)$ .

then the instruction  $(C\ 1\ 55)$  stores 55 in register  $(R\ 1)$ . Afterward, the contents of the registers are  $(17\ 55\ 671\ 3\ 81\ 95\ \dots)$ .

**INSTRUCTION 9.3** [1] *Successor Instruction*  $(S\ m)$

For each natural number  $m$ , the *successor instruction*  $(S\ m)$  adds 1 to register  $(R\ m)$ .

For example, if the current state of the registers is  $(26\ 0\ 2\ 678\ 12\ 78\ \dots)$

then the instruction  $(S\ 3)$  adds 1 to 678 so the register contents afterward are  $(26\ 0\ 2\ 679\ 12\ 78\ \dots)$ .

**INSTRUCTION 9.4** [2] *Transfer Instruction*  $(T\ m\ n)$

For natural numbers  $m, n$  the *transfer instruction*  $(T\ m\ n)$  copies the contents of register  $(R\ n)$  to register  $(R\ m)$ . All other registers are left unchanged.

If the current state of the registers is  $(17\ 0\ 2\ 679\ 3\ 81\ \dots)$  then the instruction  $(T\ 0\ 4)$  replaces  $(R\ 0)$  containing 17 with  $(R\ 4)$  which is 3 so the register contents are now  $(3\ 0\ 2\ 679\ 3\ 81\ \dots)$ .

**INSTRUCTION 9.5** [3] *Address Instruction*  $(A\ m\ n)$

For natural numbers  $m, n$  the *address instruction*  $(A\ m\ n)$  copies the contents of register  $(R\ (R\ n))$  to register  $(R\ m)$ . All other registers are left unchanged.

If the current state of the registers is  $(17\ 0\ 2\ 679\ 3\ 81\ \dots)$  then  $(R\ 4) = 3$  and  $(R\ (R\ 4)) = 679$ . Thus, instruction  $(A\ 5\ 4)$  replaces  $(R\ 5)$  containing 81 with 679. After the execution of instruction  $(A\ 5\ 4)$  the register contents are  $(17\ 0\ 2\ 679\ 3\ 679\ \dots)$ .

**INSTRUCTION 9.6** [4] *Jump Instruction*  $(J\ m\ n\ q)$

For natural numbers  $m, n, q$ , if the contents of register  $R_m$  equals the contents of register  $R_n$  i.e.  $(R\ m) = (R\ n)$ , then the *jump instruction*  $(J\ m\ n\ q)$  causes the program execution to jump to the instruction stored in  $(R\ q)$  i.e. the contents of register  $R_q$ . Otherwise, if  $(R\ m) \neq (R\ n)$  the next instruction following instruction  $(J\ m\ n\ q)$  is executed. In the special case where  $(R\ q)$  is beyond the last instruction in program  $P$ , then the execution of program  $P$  *terminates (halts)*.

**INSTRUCTION 9.7** [5] *Delete Instruction*  $(D\ m\ n)$

Starting with instruction  $(R\ m)$  delete the next  $(R\ n)$  instructions. If  $(R\ n) = 0$ , then no instructions are deleted. If  $(R\ n) + (R\ m) - 1$  points to or beyond the last instruction of the current program  $P$ , then the execution of  $(D\ m\ n)$  deletes instruction  $(R\ m)$  and all of the instructions in program  $P$  that follow instruction  $(R\ m)$ . As a consequence, execution of program  $P$  will halt if instruction  $(R\ m)$  precedes or immediately follows instruction  $(D\ m\ n)$ .

**INSTRUCTION 9.8** [6] *Update Instruction* (U k n q)

At execution time, the *update instruction* (U k n q) inserts (R n) instructions at line (R q) into program P starting at register k according to the following rules:

- The (R n) inserted instructions and their corresponding arguments are determined by (R k), (R k+1), (R k+2), . . . [in other words the contents of registers k, k+1, k+2, . . .] until (R n) instructions are completely determined.
- If (R n) = 0, then no instructions are inserted.
- The type of the inserted instruction is determined by the table of equivalence classes:

C	S	T	A	J	D	U
[0]	[1]	[2]	[3]	[4]	[5]	[6]

where  $[n] = \{m: m \text{ is a natural number and } 7 \text{ divides } m\}$

- The *contents of register k*, (R k), lies in exactly one of [0], [1], [2], [3], [4], [5], [6]. Thus, the type of instruction is unambiguously determined by the previous table.
- The 1<sup>st</sup> argument m in (C m n), (S m), (T m n), (A m n), (J m n q), (D m n), (U m n q) is determined by (R k+1) *the contents of register k+1*.
- If (R n) > 1 and it is a *Successor* instruction, then the *contents of register k+2* determine the type of the second instruction. Otherwise, the *contents of register k+2* determines the value n in one of the instructions (C m n), (T m n), (A m n), (J m n q), (D m n), (U m n q).

- If  $(R\ n) > 1$  and the type of the first inserted instruction is a *Constant, Transfer, Address* or *Delete* instruction, then the *contents of register  $k+3$*  determines the type of the second inserted instruction.
- If the type of the first inserted instruction is a *Jump* or *Update* instruction, then the *contents of register  $k+3$*  determines the value of  $q$  in the instruction  $(J\ m\ n\ q)$  or  $(U\ m\ n\ q)$ .
- If  $(R\ n) > 1$  and the first inserted instruction is *Jump* or *Update*, then the *contents of register  $k+4$*  determine the type of the second inserted instruction.
- The type and arguments of the 2<sup>nd</sup> instruction, 3<sup>rd</sup> instruction, . . . , up to the  $(R\ n)$  instruction are determined inductively according to the previous rules – using consecutive registers starting at
  - i.* Register  $k+2$  if  $(R\ k)$  lies in  $[1]$  i.e.  $(S\ m)$  is the first instruction.
  - ii.* Register  $k+3$  if  $(R\ k)$  lies in  $[0] \cup [2] \cup [3] \cup [5]$  i.e. the first instruction is  $(C\ m\ n)$ ,  $(T\ m\ n)$ ,  $(A\ m\ n)$ , or  $(D\ m\ n)$ .
  - iii.* Register  $k+4$  if  $(R\ k)$  lies in  $[4] \cup [6]$  i.e. the first instruction is  $(J\ m\ n\ q)$  or  $(U\ m\ n\ q)$ .

**DEFINITION 9.9**     *Dynamic Register Machine Program Execution*

A DRM program  $P$  is a finite sequence of instructions such that each instruction in the program is either a *Successor*, *Constant*, *Transfer*, *Address*, *Jump*, *Update*, or *Delete* instruction. Before DRM program execution begins, the value of every register is 0. In other words,  $(R_k) = 0$  for every  $k$ .

The instructions in the program  $P$  are referenced by the *Jump*, *Update*, or *Delete* instructions in the following way. At the time of dynamic register machine execution of one of these three instructions, the first instruction in  $P$  is the  $0^{\text{th}}$  instruction denoted as  $(P_0)$ . The next instruction, if it exists, in  $P$  is  $(P_1)$ . Inductively, the next instruction after  $(P_k)$ , if it exists, is  $(P_{k+1})$ .

When the program  $P$  begins execution, the  $0^{\text{th}}$  instruction  $(P_0)$  is executed first, if it exists. If at some execution step program  $P$  has no instructions, then the dynamic register machine execution of program  $P$  halts. (It is possible for a *Delete* instruction to delete the whole program or the initial program  $P$  may be empty.)

Inductively, if the  $k^{\text{th}}$  instruction,  $(P_k)$ , is being executed and it is not a *Jump*, *Update*, or *Delete* instruction, then the next instruction executed is  $(P_{k+1})$ . If the instruction  $(P_{k+1})$  does not exist, the dynamic register machine execution of program  $P$  halts.

**DEFINITION 9.10**     *Jump Execution*

If  $(P\ k)$  is being executed and  $(P\ k) = (J\ m\ n\ q)$  then  $(R\ m)$  and  $(R\ n)$  are compared.

If  $(R\ m) \neq (R\ n)$  then the next instruction executed, if it exists, is  $(P\ k+1)$ . If  $(P\ k+1)$  does not exist, then execution of program  $P$  halts.

If  $(R\ m) = (R\ n)$  then the next instruction executed, if it exists, is  $(P\ (R\ q))$ . In other words, instruction  $(P\ j)$  is executed where  $j = (R\ q)$ . If  $(P\ j)$  does not exist, then execution of program  $P$  halts.

Before presenting the formalism that describes the execution of the *Delete* and *Update* instructions, the simplest way to understand the deletion and insertion of instructions is that execution of the program  $P$  behaves like a linked list of instructions. If one or more instructions are deleted from the linked list  $P_{old}$ , then the instruction executed next is the one that immediately follows – in the sense of the new linked list  $P_{new}$  – the previous instruction  $(D\ m\ n) = (P_{old}\ k)$  executed in  $P_{old}$ . If one or more instructions are inserted into the linked list  $P_{old}$ , then the instruction executed next is the one that immediately follows – in the sense of the new linked list  $P_{new}$  – the previous instruction  $(U\ m\ n\ q) = (P_{old}\ k)$  that was executed.

A dynamic register machine example is presented first that illustrates the *linked list* behavior of the *Delete* and *Update* instructions. Then the formal rules are presented for determining the next instruction that is executed after a *Delete* or *Update* instruction.

**EXAMPLE 9.11** *Dynamic Register Machine Program with Update, Delete Instructions*

<u>Instruction</u>		
<u>Number</u>	<u>Instruction</u>	<u>Program Comments</u>
0	> (C 0 2)	Store 2 in register 0. Set (R 0) = 2.
1	(C 2 4)	Store 4 in register 2.
2	(C 5 1)	Store 1 in register 5.
3	(S 5)	Add 1 to register 5.
4	(T 6 5)	Copy contents of register 5 to register 6.
5	(A 7 5)	Copy (R (R 5)) to register 7.
6	(C 8 3)	Set (R 8) = 3.
7	(C 3 9)	Set (R 3) = 9.
8	(U 5 0 5)	Insert (R 0) instructions at (P (R 5)) with instruction translation starting at (R 5)
9	(J 2 5 2)	Jump to (R 2) if (R 2) is equal to (R 5)
10	(D 3 2)	Delete (R 2) instructions beginning at (P (R 3))
11	(C 0 17)	Set (R 0) = 17.
12	(S 0)	Add 1 to register 0.

The program begins execution with the instruction (P 0) which is (C 0 2). The third column headed by Program Comments describes the results of executing that instruction. After instruction (T 6 5), executes, the contents of the registers is (2 0 4 0 0 2 2 0 0 ...).

Before instruction (A 7 5) executes, (R 5) = 2 and (R 2) = 4. Thus, the execution of (A 7 5) sets (R 7) = 4. After instruction (A 7 5) executes, the contents of the registers are (2 0 4 0 0 2 2 4 0 0 ...).

After instruction (C 3 9) executes, the contents of the registers are (2 0 4 9 0 2 2 4 3 0 0 ...).

Thus, when the instruction (U 5 0 5) executes (R 5) = 2, (R 0) = 2, (R 6) = 2, (R 7) = 4, and (R 8) = 3. Thus, two instructions are inserted at (P 2) where they are determined starting at register 5. Since (R 5) = 2, the first inserted instruction is a *transfer* instruction. Since (R 6) = 2 and (R 7) = 4 the first inserted instruction is (T 2 4). Since register 7 completes the first inserted instruction, (R 8) = 3 determines that the second inserted instruction is an *address* instruction. The second inserted instruction is (A 0 0) because (R 9) = (R 10) = 0.

After instruction (U 5 0 5) has executed, the next instruction executed is (J 2 5 2). The table on the following page shows program P before it executes instruction (J 2 5 2) and after it has been updated by the execution of instruction (U 5 0 5):

<u>Instruction</u>	
<u>Number</u>	<u>Instruction</u>
0	(C 0 2)
1	(C 2 4)
2	(T 2 4)
3	(A 0 0)
4	(C 5 1)
5	(S 5)
6	(T 6 5)
7	(A 7 5)
8	(C 8 3)
9	(C 3 9)
10	(U 5 0 5)
11	> (J 2 5 2)
12	(D 3 2)
13	(C 0 17)
14	(S 0)

When instruction (J 2 5 2) is executed,  $4 = (R 2) \neq (R 5) = 2$  so the next instruction executed is (D 3 2). Since  $(R 2) = 4$  and  $(R 3) = 9$ , four instructions are deleted from program P starting at instruction (P 9). As a result, instructions (C 3 9), (U 5 0 5), (J 2 5 2), and (D 3 2) are deleted from program P and the current program is shown in the following table after (D 3 2) is executed:

<u>Instruction</u>	
<u>Number</u>	<u>Instruction</u>
0	(C 0 2)
1	(C 2 4)
2	(T 2 4)
3	(A 0 0)
4	(C 5 1)
5	(S 5)
6	(T 6 5)
7	(A 7 5)
8	(C 8 3)
9	> (C 0 17)
10	(S 0)

The next instruction executed is  $(C \ 0 \ 17)$ . After instruction  $(S \ 0)$  is executed, there are no more instructions that follow  $(S \ 0)$ , so the program execution halts. The final program is the same as in the previous table. When the program execution halts, the content of the registers is  $(18 \ 0 \ 4 \ 9 \ 0 \ 2 \ 2 \ 4 \ 3 \ 0 \ \dots)$

**DEFINITION 9.12**     *Delete Execution*

If  $(P_{old} \ k)$  is being executed and  $(P_{old} \ k) = (D \ m \ n)$ , then there are three cases for determining the next instruction that is executed. To define the three cases, it is first determined whether the execution of the delete command deletes itself – the  $k$ th instruction from  $P_{old}$ . The formal conditions for this to be true is that  $(R \ m) \leq k < (R \ m) + (R \ n)$ . If this condition is true, then CASE A shown below determines the next instruction that is executed.

CASE A.  $(R \ m) \leq k < (R \ m) + (R \ n)$ . After the deletion of the  $(R \ n)$  instruction(s) from program  $P_{old}$  starting at instruction  $(R \ m)$  in program  $P_{old}$ , the new program is  $P_{new}$  and the next instruction executed is  $(P_{new} \ (R \ m))$ .

CASE B.  $k < (R \ m)$ . The next instruction executed is  $(P_{new} \ k+1)$ .

CASE C.  $(R \ m) + (R \ n) \leq k$ . The next instruction executed is  $(P_{new} \ k + 1 - (R \ n))$ .

**DEFINITION 9.13**     *Update Execution*

If  $(P_{old} \ k)$  is being executed and  $(P_{old} \ k) = (U \ m \ n \ q)$ , then there are three cases for determining the next instruction that is executed. To define the three cases, before  $(U \ m \ n \ q)$  is executed, consider the instruction  $(P_{old} \ k+1)$  the instruction that immediately follows  $(P_{old} \ k)$  in program  $P_{old}$  right before the instructions are inserted.

CASE A.  $(R\ q) < k$ . After the  $(R\ n)$  instructions are inserted into program  $P_{old}$  at location  $(R\ q)$ , then instruction  $(P_{old}\ k+1)$  now in  $P_{new}$  is executed next. In other words, the next instruction executed is  $(P_{new}\ k+1+(R\ n))$ .

CASE B.  $(R\ q) = k$ . After the  $(R\ n)$  instructions are inserted into program  $P_{old}$  at location  $(R\ q)$ , then the first instruction inserted is executed next.

CASE C.  $(R\ q) > k$ . After the  $(R\ n)$  instructions are inserted into program  $P_{old}$  at location  $(R\ q)$ , then instruction  $(P_{old}\ k+1)$  is executed next. In other words, the next instruction executed is  $(P_{new}\ k+1)$ .