2.2 Computability

In the practice of computer science it is valid to question as to the generality and span of computer operations. When dealing with computer numbers we are actually approximating an infinite entity with a finite one. That is, real numbers and computer numbers are, of course, different entities. Our attempts to deal with infinity have been the source of some of the most interesting and rewarding intellectual enterprises of the century. It is relevant to provide a frame of reference that allows us to theoretically justify such an approach. It turns out that the representation of certain theoretically interesting devices called Turing Machines (TMs) is genetically straightforward and, hence, interesting in its own right.

In the following sections we make a brief introduction to the theoretical foundations of computation.

2.2.1 Recursive Functions

In the theory of computation it is customary to attempt the characterization of the meaning of what "computable" really means by postulating a set of basic functions and showing that «all» computable functions stem from the manipulation of these functions in a constructive fashion via a set of simple rules.

2.2.1.1 Basic Functions

The building blocks in the hierarchy of functions now to be defined are the so-called basic functions. The basic functions may be chosen in a variety of ways and here we select the following three:

a) The null function N(x) defined as follows:

$$N(x)=0 \qquad (2.2.1.a)$$

b) The successor function S(x) defined as:

$$S(x)= x+1 \qquad (2.2.1.b)$$

c) The unit selection function $U_i^n(x)$ defined as:

$$U_i^n(x_1,...,x_n)=x_i \qquad (2.2.1.c)$$

Our aim is to show that all computable functions may be derived from the above basic functions by applying three operations:

a) Composition
b) Primitive recursion
c) Minimalization

2.2.1.2 Composition

Composition is simply the arbitrary inclusion of a function as an argument of another function. For example, the function F(x)=x+2 may be obtained from

$$F(x)=S(S(x))$$

Any constant[1] may be obtained from the null function, by repeatedly applying the successor function. For instance, the constant K = 4 may be derived from

$$K = S(S(S(S(N(x)))))$$

In formal terms, if we are given the functions

$$\begin{cases} f(y_1,...,y_m) \\ g_1(x_1,...,x_n) \\ \quad . \\ \quad . \\ \quad . \\ g_m(x_1,...,x_n) \end{cases}$$

as members of the class being generated, then the function

$$h(x_1,...,x_n) = f(g_1(x_1,...,x_n),...,g_m(x_1,...,x_n))$$

is obtained from these given functions by the operation of composition, as shown in figure F.2.2.1.2.1.

[1]From the discussion of section 2.1. it should be clear that the type of constant (integer or real) is a matter of how we choose to interpret the binary string.

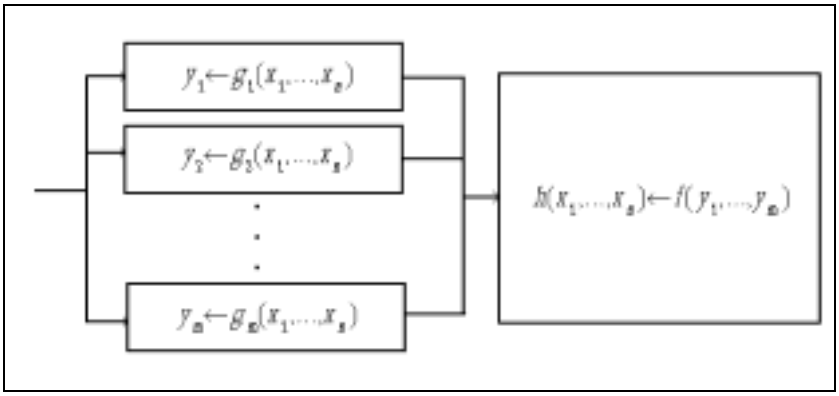


Figure F.2.2.1.2.1. Composition.

2.2.1.3 Primitive Recursion

The operation of primitive recursion is a schema (see figure F.2.2.1.3.1.) that defines a new function in terms of two given total functions. We show first how this schema defines a function of a single argument. Call the function to be defined r(x). The operation of primitive recursion includes an assignment of some specified value to r(0). It then also includes a procedure to determine r(k+1), once we have obtained r(k). We write the following equations, where k is a constant and g(x,y) is assumed to be a total function already known to be in the class.

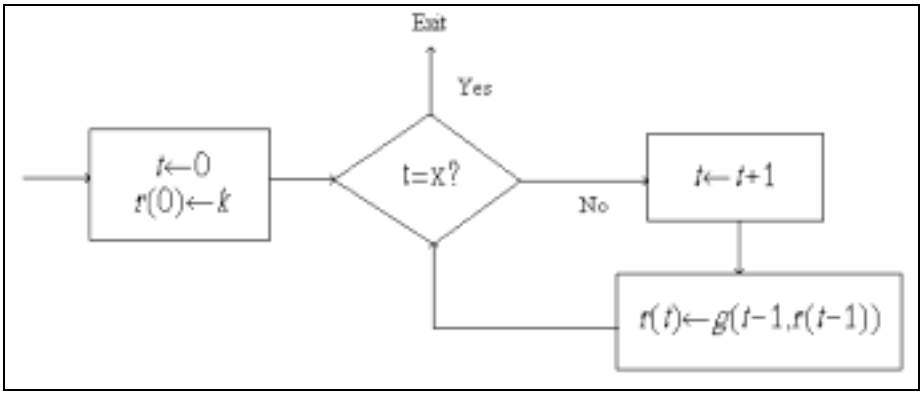


Figure F.2.2.1.3.1. Primitive Recursion.

More generally, we can define a function $r(x_1,...,x_n)$ of a number of arguments by applying this idea to some one of the arguments. Thus the following equations define $r(x_1,...,x_n)$ by giving its value for $x_1=0$ and arbitrary values to the other arguments and defining r with first argument x_1+1 in terms of the value of r with first argument x_1.

$$\begin{cases} r(0,x_2,...,x_n)=f(x_2,...,x_n) \\ r(x_1+1,x_2,...,x_n)=g(x_1,r(x_1,x_2,...,x_n),x_2,...,x_n) \end{cases}$$

The function $r(x_1,...,x_n)$ has been defined in terms of the functions $f(x_1,...,x_n), g(x_1,y,x_2,...,x_n)$

Examples of primitive recursive functions are the following:

1. $\sum(x,y)=x+y$

$$\begin{cases} \sum(0,y)=y \\ \sum(x+1,y)=S\left(\sum(x,y)\right) \end{cases}$$

2. $\Pi(x,y)=x\cdot y$

$$\begin{cases} \Pi(0,y)=0 \\ \Pi(x+1,y)=\Pi(x,y)+y=\sum(\Pi(x,y),U_2^2(x,y)) \end{cases}$$

3. $E(x,y)=x^y$

$$\begin{cases} E(x,0)=1 \\ E(x,y+1)=E(x,y)\cdot x=E(x,y)\cdot U_1^2(x,y)=\Pi(E(x,y),U_1^2(x,y)) \end{cases}$$

4. $P(x)=\begin{cases} (x-1) & for \quad x\geq 1 \\ 0 & for \quad x=0 \end{cases}$

$$\begin{cases} P(0)=0 \\ P(x+1)=x=U_1^1(x) \end{cases}$$

5. $x \dot{-} y=\begin{cases} x-y & for \quad x \geq y \\ 0 & for \quad x<y \end{cases}$

$$\begin{cases} x \dot{-} 0=x=U_1^1(x) \\ x \dot{-} (y+1)=P(x \dot{-} y) \end{cases}$$

Rich as they are, there are some calculations that primitive recursion functions may not represent. One example of these is Ackermann's function which is simply defined as:

$$A(m,n)=\begin{cases} f(0,n)=n+1 \\ f(m,0)=f(m-1,1) \\ f(m,n)=f(m-1,f(m,n-1)) \end{cases}$$

This is an example of a "doubly-recursive" function and is a special case of a class of functions defined by W. Ackermann and shown by him not to be primitive recursive.

To include functions such as Ackermann's in the set of computable functions we define a new operation.

2.2.1.4 Minimalization

Minimalization is shown, schematically, in figure F.2.2.1.4.1.

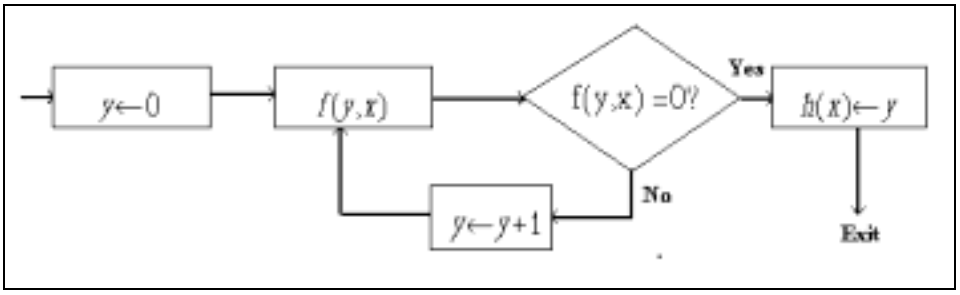


Figure F.2.2.1.4.1. Minimalization.

The inclusion of minimalization enriches our family of functions by including partial

recursive functions. The term "partial" here implies that the functions are only partially defined in some cases as shown in the following example.

Let $z=\left\lfloor \frac{x}{2} \right\rfloor$ denote the floor of x/2. We shall define $z=\left\lfloor \frac{x}{2} \right\rfloor$ by minimalization.

We have

$$\begin{aligned} z=\lfloor x/2 \rfloor &= \{ \min y \mid 2(y+1)>x \} \\ &= \{\min y \mid 2y+2 \geq x+1\} \\ &= \{\min y \mid 2y+1 \geq x\} \\ &= \{\min y \mid x \dot{-} 2y \dot{-} 1 = 0\} \end{aligned}$$

(we have used the fact that $a \geq b$ is equivalent to $b \dot{-} a = 0$).

Clearly, the existence of a terminating criterion which is evident in primitive recursive functions is a fundamental element. In partial recursive functions we are never assured that the iterations will end and this fact enriches the family of functions decisively. The relationship between the family of functions is shown in figure F.2.2.1.4.2.

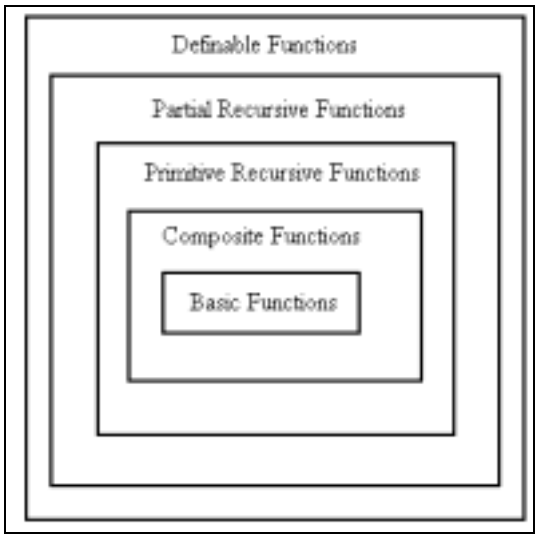


Figure F.2.2.1.4.2. A Hierarchy of Functions.

2.2.2 Church's Thesis

The reader might well ask, upon seeing figure F.2.2.1.4.2., what lies in the region named "Definable Functions". As is evident from this figure, every family of functions in a smaller frame is included in the larger ones and that, therefore, partial recursive functions

(PRFs) are actually "the" functions. And this is basically what Alonzo Church had in mind when he postulated what is now known as Church's thesis:

All effectively computable functions are partial recursive.

In so doing he made the distinction between effectively computable and definable functions. The above assertion is not provable which is why it is called a thesis. No one has ever been able to find an effectively computable function which is not partial

recursive and it remains to clarify what we wean by "effectively computable".

Which (if any) functions are definable yet not PRF (i.e. not computable)? The answer is that there are many more non-computable functions than PRFs. It is not easy, however, to define non-computable functions. One way to do this is via a device called a Turing Machine.

2.2.3 Turing Machines

The English mathematician Alan M. Turing introduced the idea of a mathematical abstraction called, after him, a Turing Machine (TM).

TMs are, simply stated, devices designed to show the barest essentials of a clearly defined (i.e. effective) procedure whereupon one is able to express the computational power of all partial recursive functions. Furthermore, if we are careful in the definition of a TM it is straightforward to establish a one-to-one correspondence between the basic functions, composition, primitive recursion and minimalization and the corresponding TM. Schematically, a TM may be depicted as shown in figure F.2.2.3.1.

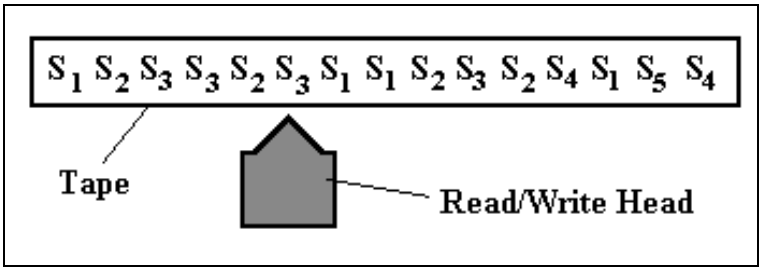


Figure F.2.2.3.1. A Turing Machine.

In such device, there is a string of symbols denoted in the figure by $S_1, S_2, ..., S_5$; these symbols are set on a tape which extends to $-\infty$ and $+\infty$ to the left and right respectively. A read/write head moves scanning the symbols on the tape and taking one of the possible following actions:

a) It writes one of the symbols of the alphabet on the tape.
b) It moves one position either to the left or right (or not at all) of the current position.

This is done one step at a time when the TM passes from one state to another. By state we mean, precisely, the status of the machine. The way in which the TM changes from state to state is defined by a control program (or control unit) such as the one shown in table T.2.2.3.1.

Table T.2.2.3.1. The Control Unit of a Turing Machine

Present Sate	Input = 0			Input = 1		
	New Symbol	Head Movement	Next Sate	New Symbol	Head Movement	Next State
Q1	1	RIGHT	Q2	1	LEFT	Q3
Q2	0	LEFT	Q1	0	LEFT	Q4
Q3	1	LEFT	Q1	1	LEFT	HALT
Q4	1	LEFT	Q2	1	RIGHT	Q5
Q5	0	RIGHT	Q4	0	RIGHT	Q2

The particular TM whose control unit is shown is assumed to operate with a binary alphabet and on a tape which is initially full of 0's. Accordingly, there are two columns: the first one corresponds to the actions the machine is to take when the symbol under the head is "0".

Likewise, the second column corresponds to the actions it is to take when the symbol being read is a "1". Notice that state Q3, when presented with a "1" in its tape, leads to a state labeled "HALT". This special state signifies that the TM is to stop all activity. It is possible for the TM in question to enter this halting state from several other states or none at all. One interesting problem is to find out which TM, given a fixed number of states, is able to write the largest number of ones on a tape initially filled with zeroes[2]. The reader is invited to test the above TM with the program TURIMACH.EXE included in the diskette which complements this text.

It is common to talk of a TM when referring to a particular TM's control unit and, indeed, the actual internal working of any TM depends exclusively on its control unit.

[2]The maximum number of 1's written on the tape by the TM is called its productivity. The function denoting the productivity of an n-state TM is denoted by $\sigma(n)$ and is known as Rado's function.

Unless stated otherwise, we shall adhere to that convention from now on and simply speak of a TM when referring to its control unit.

It should be clear from the discussion above that no TM is physically realizable since the tape is infinite. The best we can do is to simulate a TM with a finite (albeit arbitrarily long) tape. The main interest in these "devices" is, as stressed in the opening paragraphs, that a one to one correspondence between TMs and computable functions is possible.

2.2.3.1 Normal Turing Machines

To clearly expose the relationship between PRFs and TMs we shall define a set of conventions which lead us to what is called a *normal* TM. The conventions are the following:

1. An input argument, *m*, will be represented on the input tape as a string of m+1 1's. Letting $\overline{m}$ be the machine tape representation of *m* we have:

$$\overline{5} = 111111$$
$$\overline{0} = 1$$

2. An *n*-tuple of input arguments will be represented on the input tape as an *n*-tuple of the representations of the individual arguments, separated by intervening blanks[3]. That is,

$$\overline{(m_1,m_2,\ldots,m_n)} = \overline{m_1} B \overline{m_2} B \ldots B \overline{m_n}$$

3. We shall assume that a machine starts in state Q1 with the head over the leftmost 1.
4. We shall also assume that the final head position is over the leftmost 1 on the tape.
5. We shall employ special end-of-data marks to signal the two squares at the end of the data. We shall use the asterisk **(*)** for this purpose.
6. If a machine does not stop after starting on some input tape, the partial function determined by the machine will be undefined for the input argument(s) represented on the tape.

2.2.3.2 Turing's Thesis

Now we may state what is called Turing's Thesis:

---

[3]For the purpose of the discussion we assume that a *0* on the tape is equivalent to a *blank* square.

If a function is partial recursive, it can be computed on some (regular) Turing Machine.

Given Church's Thesis, this last thesis may be re-phrased as:

Any effectively computable function may be calculated by a Turing Machine.

2.2.4 TM-Computable is Effectively Computable

A one-to-one correspondence between what is computable and what is TM-computable may be established along the following lines:

a) Show that the basic functions may be computed by some TM.
b) Show that it is possible to construct new functions by using composition, primitive recursion and minimalization. These will be TM-computable

At this point we will have shown that any PRF may be calculated by a TM. Then,

c) Show that all functions calculated by a TM are PRFs.

Once having shown these, the conclusion is that any computable function is computable by a TM and that, furthermore, if it cannot be computed by a TM it can not be computed at all.

In what follows we expose the plausibility of, rather than prove, the stated assertions. We do this by offering to the reader a set of examples along the lines mentioned above.

We shall use a more compact description of a TM by defining it as a set of quintuples of the form:

Qi, Ii, Oi, M, Qj

where

Qi ≙ present state Ii ≙ input symbol
Oi ≙ output symbol M ≙ head movement
Qj ≙ next state

2.2.4.1 The Basic Functions are TM-Computable

A TM which performs as the null function [N(x)] is given next.

N(x) : TM{N(x)}

```
Q0,1,1,R,Q0        ; MOVE ALL THE WAY TO THE RIGHT
Q0,*,0,L,Q1        ; ERASE RIGHT *
```

Q1,1,0,L,Q1 ; ERASE 1'S
Q1,*,*,R,Q2 ; UNTIL END OF DATA
Q2,0,1,R,Q3 ; MAKE $\overline{0}$
Q3,0,*,L,QΩ[4] ; WRITE * AND STOP

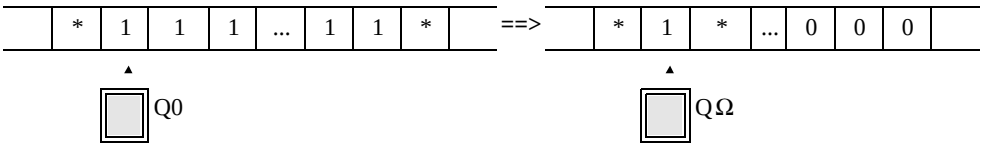


Figure F.2.2.4.1.1.

The state of the tape upon which TM{N(x)} operates is shown in figure F.2.2.4.1.1. at the beginning and the end of the computation.

A TM which performs as the successor function [S(x)] is given next.

$S(x) : TM\{S(x)_R\}$

Q0,1,1,R,Q0; Q0,*,1,R,Q0; Q0,0,*,L,Q1
Q1,1,1,L,Q1; Q1,*,*,R,QΩ

This version of the successor function adds a 1 to the right of the original argument. Hence the subscript "R".

$S(x) : TM\{S(x)_L\}$

Q0,1,1,L,Q0; Q0,*,1,L,Q0
Q0,0,*,R,QΩ

This version of the successor function adds a 1 to the left of the original argument. Hence the subscript "L".

A TM which performs as the unit selection function [$U_i^n(x_1,...,x_n)$] is given next. To this effect we assume that the arguments $x_1,...,x_n$ for $U_i^n(x_1,...,x_n)$ are set as shown in figure F.2.2.4.1.2.

[4] QΩ denotes a halting state.

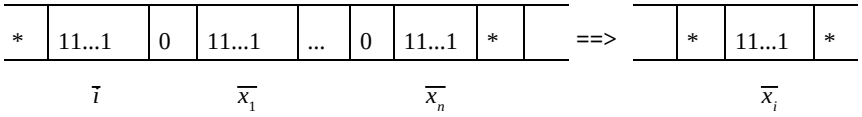


Figure F.2.2.4.1.2.

$U_i^n(x_1, x_2, ..., x_n)$: TM{ $U_i^n(x_i)$ }

```
Q1,1,1,L,Q1; Q1,*,0,R,Q2; Q2,1,0,R,Q3          ; ERASE * AND FIRST "1"
Q3,1,0,R,Q4                                    ; ERASE A "1" (DECR. COUNTER)
Q4,1,1,R,Q5; Q4,0,0,R,Q6                       ; SEE IF COUNT=0 (Q6)
Q6,0,0,R,Q6, Q6,1,1,L,Q7                       ; FIND THE ARGUMENT
Q7,0,*,R,Q7; Q7,1,1,R,Q8                       ; MARK END OF DATA
Q8,1,1,R,Q8; Q8,*,*,L,Q12; Q8,0,*,R,Q10        ; LAST ARGUMENT OR NOT
Q10,1,0,R,Q10; Q10,0,0,R,Q10 ; Q10,*,0,L,Q11   ; CLEAR REST OF TAPE
Q11,0,0,L,Q11; Q11,*,*,L,Q12; Q12,1,1,L,Q12    ; TRAVERSE ARGUMENT
Q12,*,*,R,QΩ                                   ; AND STOP
Q5,1,1,R,Q5; Q5,0,0,R,Q13                      ; TRAVERSE COUNTER
Q13,0,0,R,Q13; Q13,1,0,R,Q14; Q14,1,0,R,Q14    ; ERASE ARGUMENT
Q14,0,0,L,Q15; Q15,0,0,L,Q15; Q15,1,1,L,Q16    ; FIND COUNTER AGAIN
Q16,1,1,L,Q16; Q16,0,0,R,Q3                    ;RE-START CYCLE
```

2.2.4.2 Composition is TM-Computable

Since normal TMs start and stop on the leftmost '1', composition is a matter of simply re-labeling the ending state QΩ and joining the resulting TMs. For example, take the following function:

$$F(x)=1 \implies F(x)=S(N(x))$$

$$\left.\begin{array}{l} \left.\begin{array}{l} Q10,1,1,R,Q10;Q10,*,0,L,Q11;Q11,1,0,L,Q11 \\ Q11,*,*,R,Q12;Q12,0,1,R,Q13;Q13,0,*,L,Q20 \end{array}\right\} N(x) \\ \left.\begin{array}{l} Q20,1,1,R,Q20;Q20,*,1,R,Q21;Q21,0,*,L,Q21 \\ Q21,1,1,L,Q21;Q21,*,*,R,Q\Omega \end{array}\right\} S(x)_R \end{array}\right\} S(N(x))$$

2.2.4.3 Primitive Recursion is TM-Computable

In primitive recursion we have two basic arguments:

a) The variable in *y* (initial value "k").
b) The free variable in *x*.

We shall assume *x* and *y* to be the two arguments in a TM. By convention *y* will be leftmost and "grow" leftwise; the counter "t" will be rightmost and grow rightwise. (See figure F.2.1.4.3.1).

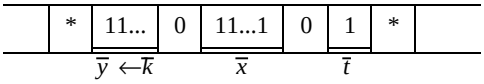


Figure F.2.1.4.3.1. Tape Configuration.

We need a "subroutine" to compare "t" with *x*. This may be accomplished as follows:

TM{CP} : *Verify if x=t on a k:x:t tape*

```
Q1,1,1,R,Q1; Q1,0,0,R,Q2                    ; TRAVERSE k̄
Q2,1,1,R,Q2; Q2,0,0,R,Q3                    ; TRAVERSE x̄
Q3,1,1,R,Q3; Q3,*,*,L,Q4; Q3,#[5],#,L,Q4    ; TRAVERSE t̄
Q4,1,#,L,Q5; Q4,0,0,L,Q8                    ; Q5 -> "t" REMAINS
                                            ; Q8 -> "t" IS EXHAUSTED
Q5,1,1,L,Q5; Q5,0,0,L,Q6                    ; BACKTRACK "t"
Q6,1,1,L,Q6; Q6,0,0,R,Q7; Q6,#,#,R,Q7       ; BACKTRACK x
Q7,1,#,R,Q2                                 ; START NEXT PASS
Q8,1,1,L,Q8; Q8,#,#,R,Q9                    ; RETRACE WHEN "t" IS
                                            ; EXHAUSTED
Q9,0,0,R,Q10; Q9,1,1,R,Q12                  ; Q10: x AND "t" ARE EQUAL
Q10,0,0,R,Q10; Q10,1,1,R,Q10;               ; Q12: x AND "t" ARE NOT EQUAL
Q10,#,#,R,Q10; Q10,*,*,L,Q11                ; GO TO EOD WHEN EQUAL
Q11,#,1,L,Q11; Q11,0,0,L,Q11;
Q11,1,1,L,Q11; Q11,*,*,R,Q_YES              ; x AND "t" ARE EQUAL
Q12,0,0,R,Q12; Q12,1,1,R,Q12;
Q12,#,#,R,Q12; Q12,*,*,L,Q13                ; GO TO EOD WHEN NOT EQUAL
Q13,0,0,L,Q13; Q13,1,1,L,Q13;
Q13,#,1,L,Q13; Q13,*,*,R,Q_NOT              ; x AND "t" ARE NOT EQUAL
```

The TM just described may be found in the file CMPX_T.NTM. The reader should simulate this TM and watch the way it works.

Let us utilize the tools already developed to obtain

[5]We take advantage of an auxiliary symbol "#" to facilitate matters. Hence, these particular TMs have a quaternary alphabet.

$$\begin{cases} \sum(x,y) = x+y \\ \sum(0,y) = y \\ \sum(x+1,y) = S(\sum(x.y)) \end{cases}$$

We use TM{$S(x)_L$} to conform to our conventions and calculate $\sum(4,3)$. The initial tape is

$$* \quad \underbrace{1111}_{\sum(0,y)} \quad 0 \quad \underbrace{11111}_{\bar{x}} \quad 0 \quad \underbrace{1}_{\bar{t}} \quad *$$

Make:

$$Q_{YES} \triangleq Q22$$
$$Q_{NOT} \triangleq Q15$$

and use the previously defined TMs, thus:

1) Compare *x* and "t"

<TM(CP)>

2) Increment the partial result $\sum$.

<$S(x)_L$>
Q15,1,1,L,Q15; Q15,*,1,L,Q15; Q15,0,*,R,Q16 ;

3) Now increment $\bar{t}$ and iterate:

This may be accomplished as follows:

TM{NXT} : *Increment* $\bar{t}$

```
Q16[6],1,1,R,Q16; Q16,0,0,R,Q17      ; TRAVERSE y
Q17,1,1,R,Q17; Q17,0,0,R,Q18         ; TRAVERSE x
Q18,1,1,R,Q18; Q18,*,1,R,Q18;
Q18,0,*,L,Q19                        ; S(x)R - "t"<-"t"+1
Q19,1,1,L,Q19; Q19,0,0,L,Q20         ; BACK ON "t"
Q20,1,1,L,Q20; Q20,0,0,L,Q21         ; BACK ON x
```

[6]We have started numbering the states with "16" to ease the composition labeling which follows.

```
Q21,1,1,L,Q21; Q21,*,*,R,Q1[7]          ; NEW ITERATION
```

4) Now we eliminate the gaps in the tape in order to conform to the normal standard, where the result is expected in a single block of ones, with the head on the leftmost "1". This we accomplish, thus:

```
Q22,0,0,R,Q22; Q22,1,1,R,Q22; Q22,*,0,L,Q23      ; TRAVERSE THE DATA
Q23,0,0,L,Q24; Q23,1,0,L,Q23                     ; ELIMINATE "t"
Q24,1,0,L,Q24; Q24,0,*,L,Q25                     ; ELIMINATE x
Q25,1,1,L,Q25; Q25,*,*,R,HALT                    ; ADOPT NORMAL POSITION
```

The TM that calculates $\sum(x,y)$ may be found in the file SUMX_Y.NTM. The reader should simulate this TM and watch the way it works. Simply set the tape as indicated above and load (or directly key in the control unit) and see a partial recursive function's direct simulation.

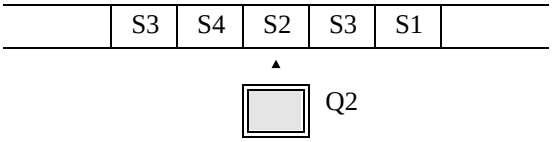
The reader should have no trouble in convincing himself that:

a) It is simple to extend the method above to include minimalization.
b) The appropriate application of the corresponding TMs will necessarily yield the desired partial recursive functions.

Once the above is clear, it immediately follows that **any** PRF is representable by a TM as the previous example illustrated.

2.2.5 Effectively Computable is TM-Computable

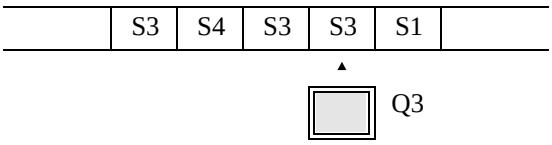
We have given reasonable arguments to support the view that any PRF is TM-computable. But, are all TM-computable functions PRFs? We now argue that, in effect, such is the case. First we define an instantaneous description (ID) as the one which allows us to determine the state of a TM. For instance, let us assume that we have the following tape:



[7] By Q1 we designate the first step in the process of iteration.

In this tape we have all the elements which allow us to determine the status of the computation at a particular time. We may think of such description as a number. Using the symbols 0,1,...,9,Q,S,R,N,L[8] each of these may be thought of as a base 15 number. Then we may represent any computation as the set of all transitions from an initial to a final state. In other words, from an ID to another ID. The functions defining the transitions from ID_i to ID_{i+1} constitute the TM-computable functions.

In the tape above, we let its ID to be S3S4Q2S2S3S1. Our convention, therefore, is that we shall represent the state the machine is in, as well as the position of the head, by including the state number (in this case Q2) immediately before the symbol written on the tape in the position the head is about to read. Let us assume that, by virtue of the TM's quintuples the tape is transformed into



whose ID is

S3S4S3Q3S3S1

Therefore, we may think of this operation as the one resulting from the "application" of an operator that transforms the base 15 number

$$N_1 = \text{S3S4Q2S2S3S1}$$

into the base 15 number

$$N_2 = \text{S3S4S3Q3S3S1}$$

We may put

$$N_2 = f(N_1)$$

This function may be given by

$$d=\{\min\ k \mid R([\lfloor N_1/15^k \rfloor - Q2S2],15^4)=0\}$$

where R(x,y) ≙ residue of x/y.

Here, d = number of symbols after Q2S2 (d = 4).

[8]The symbols "R", "L" and "N" stand for "Right", "Left" or "None" and these clearly refer to the allowed movements of the head. That is, we allow for the head not to move from its present position in some cases.

From the ID's above we have:

$\lfloor N_1/15^4 \rfloor$ -Q2S2 = S3S4Q2S2-Q2S2 = S3S40000

and

$R(S3S40000,15^4) = 0$

Then:

$$N_2 = 15^{d-4} \lfloor N_1 / 15^{d-4} \rfloor + 15^d(S3Q3) + R(N_1, 15^d)$$

Making

$\alpha = 15^8 \lfloor N_1/15^8 \rfloor$ = S3S400000000

$\beta = 15^4 \times$ S3Q3 = S3Q30000

$\gamma = R(N_1,15^4)$ = S3S1

$N_2 = \alpha+\beta+\gamma$ = S3S4S3Q3S3S1

Which is, indeed, the required ID. □

There are two basic kinds of quintuples in a TM: stopping and non-stopping, which we denote as follows:

non-stopping: $M_1,...,M_p$

stopping: $P_1,...,P_m$

Let us define a set of functions $g_1,...,g_p$ and $f_1,...,f_m$ corresponding to the non-stopping and stopping quintuples, respectively.

As an example, let us analyze the following

TM{$S(x)_L$}: Q1,1,1,L,Q1; Q1,*,1,L,Q1; Q1,0,*,R,Q2

Q2 is a halting state.

Let us make:

S0 --> 0

S1 --> 1

S2 --> *

Then:

M_1 = Q1S1S1LQ1 ==> g_1

M_2 = Q1S2S1LQ1 ==> g_2

P_1 = Q1S0S2RQ2 ==> f_1

For example, assume:

ID_1 = S0S2S1Q1S1S1S2S0

and

ID_2 = S0S2Q1S1S1S1S2S0

Hence:

$$d=\{\min\ k \mid R(\lfloor N_1/15^k \rfloor - \left\{\begin{matrix} Q1S1 \\ Q1S2 \\ Q1S0 \end{matrix}\right\},15^4)=0\}$$

$$P=15^{d-6}\lfloor N_1/15^{d-6}\rfloor + \left\{\begin{matrix} Q1 \\ Q1 \\ Q2 \end{matrix}\right\}\times 15^{d-4} + R(\lfloor N_1/15^{d-4}\rfloor,15^2)\times 15^{d-2} + \left\{\begin{matrix} S1 \\ S1 \\ S2 \end{matrix}\right\}\times 15^{d} + R(N_1,15^d)$$

We have, for the quintuple (Q1,1,1,L,Q1)

$$d_1=\{\min\ k \mid R(\lfloor N_1/15^k \rfloor - Q1S1,15^4)=0\}$$

Here, for k=6 (d_1=6):

i) $\lfloor N_1/15^6 \rfloor$ = S0S2S1Q1S1
ii) S0S2S1Q1S1 - Q1S1 = S0S2S10000
iii) R(S0S2S10000,15^4) = 0

$$g_1=\underbrace{15^{d_1-6}\lfloor N_1/15^{d_1-6}\rfloor}_{\alpha}+\underbrace{15^{d_1-4}Q1}_{\beta}+\underbrace{15^{d_1-2}(R\lfloor N_1/15^{d_1-4}\rfloor,15^2)}_{\gamma}+\underbrace{S1\times 15^{d_1}}_{\delta}+\underbrace{R(N_1,15^{d_1})}_{\varepsilon}$$

$\alpha=15^{12}\lfloor N_1/15^{12}\rfloor$ = *S0S2000000000000*
$\beta=15^{10}Q1$ = *Q10000000000*
$\gamma=15^8(R\lfloor N_1/15^{10}\rfloor,15^2)$ = *S100000000*
$\delta=S1000000$ = *S1000000*
$\varepsilon=R(N_1,15^6)$ = *S1S2S0*

$\alpha+\beta+\gamma+\delta+\varepsilon$ = *S0S2Q1S1S1S1S2S0*

Which is, indeed, ID2. □

Let $f_1 \bigcup f_2$ mean *a function f_1 when $f_1(x)$ is defined and a function f_2 when $f_2(x)$ is*

defined[9].

We then have:

$$f \triangleq f_1 \bigcup f_2 \bigcup \cdots \bigcup f_p$$
$$g \triangleq g_1 \bigcup g_2 \bigcup \cdots \bigcup g_m$$

Since *f* is a stopping condition, we have,

$$P(x) = f\, g^*$$

where

$$g^2 = g(g(x))$$
$$g^3 = g(g(g(x)))$$
$$\vdots$$

Therefore,

$$P(x) = f \bigcup fg \bigcup fg^2 \bigcup \cdots \bigcup fg^n$$

(1) If f(x) is defined, then P(x) = f(x) and STOP.
(2) If P(x) = g(x) then x = g(x) and GOTO (1).

Now, since g_1, g_2, ... are PRF => g is PRF
since f_1, f_2, ... are PRF => f is PRF.
$\therefore$
since f and g are PRF => P(x) is also PRF.

Therefore, all the functions computed by a TM are partial recursive. □

2.2.6 Incomputable Functions

We remarked earlier that one of the purposes of defining a TM is to determine what is or is not computable. In figure F.2.2.1.4.2. we showed a set of "Definable" but not "Partial Recursive" functions. From the last paragraphs we must conclude that "Not Partially Recursive" is equivalent to "Incomputable". In what follows we give a simple example of an incomputable function and we do this, precisely, by using the concept of a TM.

It should be clear that we may count the number of all possible TMs given a fixed number of states (n), the number of symbols in the alphabet (s) and the possible head

[9]Notice that f_1 and f_2 cannot both be defined for a given argument since our TMs are deterministic.

movements (m). For example, if n=1, s=2 and m=2 there are $\tau=[(n+1)\times s\times m]^{sn}$ = $(2\times 2\times 2)^2$ = 64 possible TMs[10]. Likewise, if n=10 then $\tau=44^{20}\approx 7.4\times 10^{32}$. On the other hand, we may easily define a procedure which allows us to automatically produce all possible 1 state TMs, then all possible 2 state TMs, etc. If we accept the possibility of such automation it then follows that there is an effective and well defined procedure to label (or *number*) all TMs. Therefore, we may unambiguously speak of the i-th TM. To each TM there corresponds a PRF and, therefore, we may also number all possible computable PRFs. In other words, the set of all computable functions is infinitely denumerable.

We may establish the following correspondence:

T_1 ---> F_1
T_2 ---> F_2
. . .
T_n ---> F_n

We may now define the following function:

$$F_m(X) = F_x(X) + 1$$

For instance,

$$F_m(1) = F_1(1) + 1$$

or

$$F_m(100) = F_{100}(100)+1$$

The question is the following: To which of the PRFs (TMs) in our infinite list does $F_m(X)$ correspond? It turns out that $F_m(X)$ is *not* in the list. For assume it were. Then we could pick as the argument X the value m, and thus:

$$X = m$$

Then, by definition:

$$F_m(m) = F_m(m)+1$$

and, it follows that

$$0 = 1$$

---

[10]We allow for a halting state. Therefore, all the n-state machines are actually (n+1)-state machines.

Clearly such a contradiction implies that $F_m(X)$ is not in our infinite list of computable functions. Such being the case we have an example of a well defined but incomputable function. It should be clear that the possible variations of functions similar to $F_m(X)$ are themselves infinitely denumerable. Furthermore, if we were to number the first order incomputable functions (which we may easily do) then we could apply the same argument to the resulting list of incomputable functions. This process may be repeated an infinite number of times. Clearly, the set of incomputable functions is not denumerable and, as stated before, there are many more incomputable than computable functions. □

EXERCISES

2.6 Let *f(x)* be defined by:

$$f(x) = \begin{cases} x^2 & \text{if } x \text{ is even} \\ x + 1 & \text{if } x \text{ is odd} \end{cases}$$

Show that *f(x)* is primitive recursive.

2.7 a) Show that $y = \lfloor \sqrt{x} \rfloor$ is recursive.
b) Show that $y = \lfloor x/3 \rfloor$ is recursive.

2.8 Let

$$f(x) = \begin{cases} f(0) = 1 \\ f(1) = 3 \\ f(x+2) = (x+2) \cdot f(x) \text{ for } x > 1 \end{cases}$$

Show that *f(x)* is recursive.

2.9 Assume a stored program computer with only one register, an accumulator, and only the following machine operations:

TR + A: Transfer to location A if the contents of the accumulator are positive.
ADD A: Add the contents of location A to those of the accumulator.
STO A: Store the contents of the accumulator in location A.

Assume that the store is unlimited and that the word size is sufficiently large. Also, assume that the values 0, +1 and -1 are permanently available

in memory.

Show that the device just defined is able to perform any operation whatsoever.

2.10 Show that the "sequence", "if <> then <> else" and the "do while <> end" operations of structured programming are definable in terms of composition, primitive recursion and minimalization.

2.11 Assume a TM has a tape initially filled with 0's. The control unit which operates on such tape and fills up the tape with as many 1's as possible **and** stops is called the "most productive" TM. For example, TM = {Q1,0,1,R,Q2; Q1,1,1,L,Q2; Q2,0,1,L,Q1; Q2,1,1,R,Halt} is the most productive 2-state machine[11]. The function $\sigma(n)$= *the productivity of the most productive n-state machine* is known as Rado's function, e.g. $\sigma(2) = 4$. Show that $\sigma(n)$is incomputable.

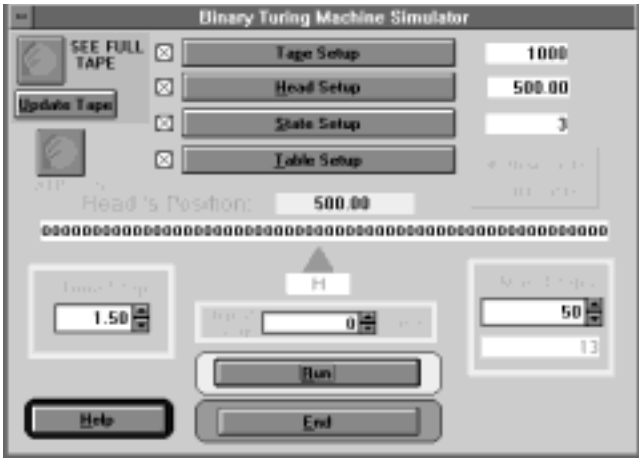
If you have not already done so, install the software of the companion diskette in your computer.

From TM Simulator's menu, select Binary Turing Machine.

2.12 1) Select *TapeSetup* and *Initialize Tape*. Set the tape size to 1000. The tape will be set to all zeros.

2) Select *Head Setup*. There you will indicate the simulator where the head of the TM will be positioned on entry. Set it to the middle. The head will be on square 500.

[11]Only *non-halting* states are considered.



3) Select *State Setup*. Set it to "3". This machine will have 3 non-halting states. Allow the *State Table Remains Visible* to remain set.

4) Select *Table Setup*. Activate *Yes* when asked if you want to retrieve a TM from a disk file.

Here you should select the file "bb3.btm". The extension "btm" stands for "Binary Turing Machine".

(Select *No* when asked if you want to store the Control Unit of the Turing Machine.)

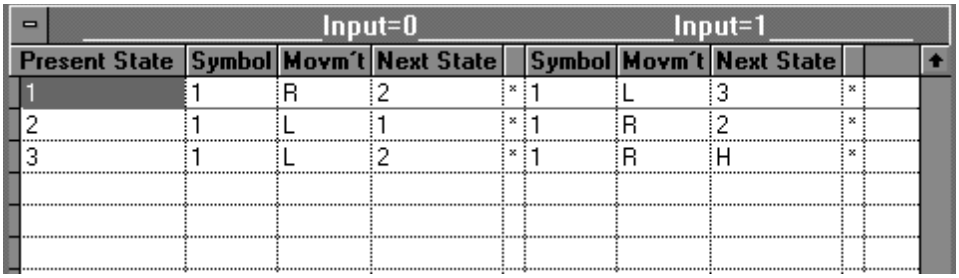
The TM you are about to simulate has already been stored in a file. The browse window which now appears displays the structure of a three-state TM. Notice the **H** on the lower right of the window. This is the *halt* state entry.

TABLE SETUP

- A browse window will appear when activating this button. Initially all entries in the table are set to '*'. They should be set to 0/1 in the case of the symbol to be written on the tape; R/L in the case of the movement the head is to perform; 1,...,n in the case of the next state the TM is to enter. The next state entries are checked to insure a valid

range. Alternatively you may specify a state denoted by 'H' to signify a HALT condition.

- If you desire to modify the contents of the table WITHOUT RE-INITIALIZING THE TABLE you must select the "Old Table" option to the right of the "Table Setup" button. Otherwise, the table will be reset completely.



	Input=0				Input=1			
Present State	Symbol	Movm't	Next State		Symbol	Movm't	Next State	
1	1	R	2	*	1	L	3	*
2	1	L	1	*	1	R	2	*
3	1	L	2	*	1	R	H	*

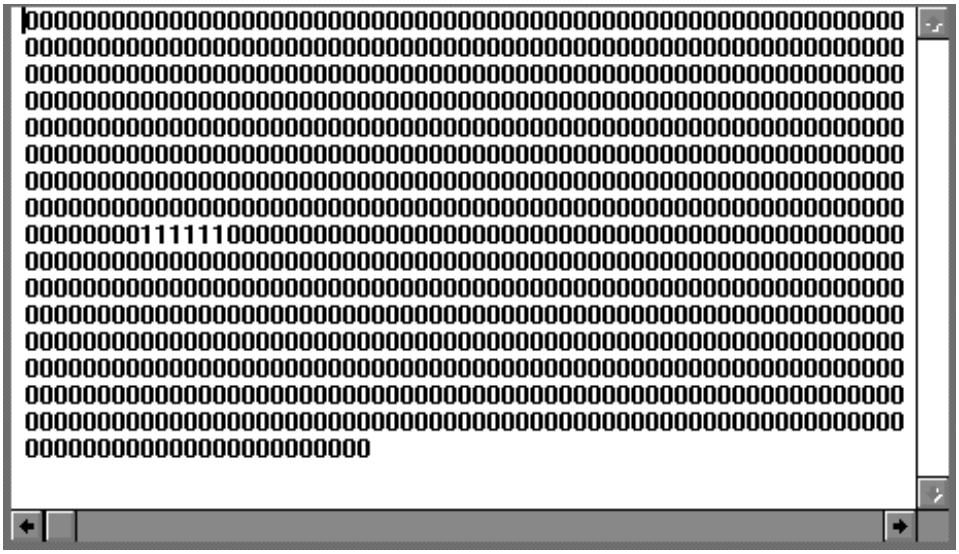
5) In *Time Step* select 1.5. This sets the machine to wait 1.5 seconds between every state transition.

6) In *Max. Steps* select 50. This is the maximum number of state transitions the simulator will undergo before the simulation stops.

7) Activate the *Run* button. This starts the simulation.

Watch the simulator as it runs. The Tape Window and the Head's Position Window are updated as the simulation advances.

8) Activate the *SEE FULL TAPE* button. Notice the pattern of 1's that appears. Press *ESC* to close this window.



9) Activate the *Statistics* button. This will get the window shown next. Notice that the number of head moves $\nu(n)$ is larger than the productivity $\sigma(n)$ and that the maximum distance the head reaches (relative to the initial position) $\delta(n)$ is smaller.

2.13 Show that $\nu(n)$ is larger than or equal to $\sigma(n)$ i.e. $\nu(n) \geq \sigma(n) \ \forall n$

2.14 Show that $\nu(n)$ is incomputable.

2.15 Activate the *See Full Tape* button. Modify some of the values of the tape so that it will no longer be filled with 0's. Redo exercise 2.12. What are the values of $\nu(n)$ $\sigma(n)$ and $\delta(n)$? Notice that if you want to temporarily stop the run you should simply press the *ESC* key.

You may now remove the window to observe the simulation's displays. Select *Yes* if you want to continue; *No* to stop the simulation.

2.16 In 1983, at the University of Dortmund, Germany, a contest was held to search for $\sigma(5)$ The winner was a TM discovered by Uwe Schult. Repeat exercise 2.12. but select the file "Schult.Btm" instead. Set the tape size to 5000; position the head on location 2500; set *Time Step* to 0; set *Report every ... steps* to 5000; set *Max. Steps* to 2,000,000. Run the simulator.

What are the values of $\nu(n)$ $\sigma(n)$and $\delta(n)$?

2.17 In December, 1984, George Uhing discovered a more productive 5-state TM. Repeat exercise 2.16 but select the file "Uhing.Btm" instead. Now set *Report every ... steps* to 10,000; set *Max. Steps* to 2,500,000. What are the values of $\nu(n)$ $\sigma(n)$and $\delta(n)$?

2.18 *From TM Simulator's menu, select NORMAL TURING MACHINE.*

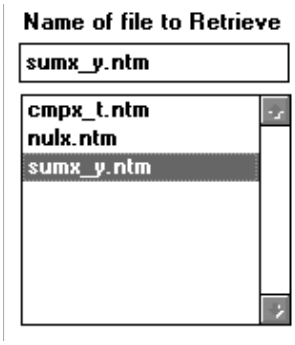
1) Activate the *Tape Setup* button; activate the *Initialize Tape* button and include (at any point roughly on the middle of the tape) the sequence ***111111011111101***; activate the *Accept Tape* button.

Activate the *End* button.

2) Activate *Head Setup*. When prompted for the head's position, activate *Yes.* When prompted for a specific position of the tape activate *End*. Unless you want the tape on a different position, the head should be positioned immediately after the leftmost asterisk. The contents of the tape at that point should conform to the Normal TM's conventions, i.e. it should be a "1". If it is not so, you will not be allowed to execute.

3) Activate the *State Setup* button. Simply activate *End*. The number of

states of the TM will be automatically set if you select a file from disk.



4) Activate *State Setup* and indicate that the TM's control unit should be retrieved from a file. Select the file *SumX_Y.Ntm*.

	Input=0				Input=1				Input=*				Input=#		
Present State	S1	Mov	N.S.		S2	Mov	N.S.		S3	Mov	N.S.		S4	Mov	N.S.
1	0	R	2	*	1	R	1	*	-	-	-	*	-	-	-
2	0	R	3	*	1	R	2	*	-	-	-	*	-	-	-
3	-	-	-	*	1	R	3	*	*	L	4	*	#	L	4
4	0	L	8	*	#	L	5	*	-	-	-	*	-	-	-
5	0	L	6	*	1	L	5	*	-	-	-	*	-	-	-
6	0	R	7	*	1	L	6	*	-	-	-	*	#	R	7
7	-	-	-	*	#	R	2	*	-	-	-	*	-	-	-
8	-	-	-	*	1	L	8	*	-	-	-	*	#	R	9
9	0	R	10	*	1	R	12	*	-	-	-	*	-	-	-
10	0	R	10	*	1	R	10	*	*	L	11	*	#	R	10
11	0	L	11	*	1	L	11	*	*	R	22	*	1	L	11
12	0	R	12	*	1	R	12	*	*	L	13	*	#	R	12
13	0	L	13	*	1	L	13	*	*	R	15	*	1	L	13
14	-	-	-	*	-	-	-	*	-	-	-	*	-	-	-

The State Table appears on a browse window. Notice that, in this case,the number of states does not allow a full view. You may browse the table activating the slide button on the right.

5) *Run* the machine(Fig. E.II.1). Set the parameters as shown in the figure.

The simulated machine illustrates the way a function (in this case the

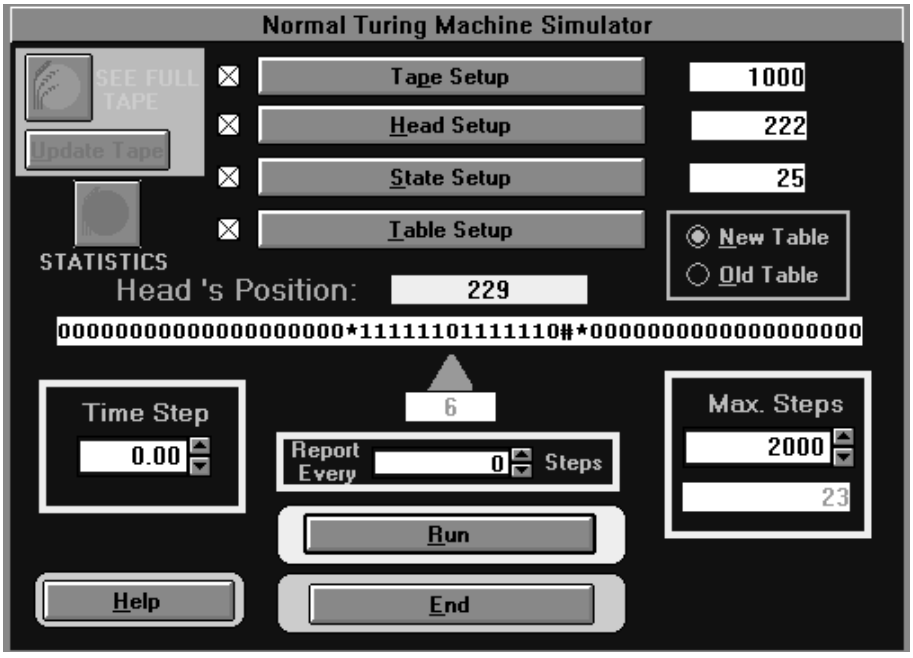


Figure E.II.1

Sum of two numbers) may be achieved from composition and primitive recursion.

2.19 The function $X + Y$ illustrated by the last example can easily be modified to make it more efficient. Find a TM that implements such function with a minimum number of states.

2.20 Find the TM which implements the product of two arguments following the primitive recursive scheme outlined in exercise 2.18:

$\Pi(x,y)=x\cdot y$

$$\begin{cases} \Pi(0,y)=0 \\ \Pi(x+1,y)=\Pi(x,y)+y=\sum(\Pi(x,y),U_2^2(x,y)) \end{cases}$$