CA320 - Computability & Complexity

Models of Computation - Turing Machines

Dr. David Sinclair

The Search for a General Model of Computation

The language $SimplPal = \{xcx^r | x \in \{a, b\}\}$ cannot be accepted by a finite automaton but it can be accepted by a push-down automaton (PDA).

A PDA cannot accept either $AnBnCn = \{a^n b^n c^n | n \in \mathcal{N}\}$ or $L = \{xcx | x \in \{a, b\}\}$.

- A stack is not sufficient.
- A finite automaton with *2 stacks* could accept $AnBnCn$.
- A finite automaton with a *queue* could accept L.

Either adding a queue or 2 stacks to a finite automaton significantly enhances the computational power of the device and either could be the bases for a general model of computation.

Alan Turing

Alan Turing (1912-1954) proposed an abstract model of computation in 1936. Though his model was a purely theoretical one, its principles and features predated many of the features of modern computers.

Turing's model was developed by considering how "human computers" at the time worked with paper and pencil.

- The data written on the paper are symbols from a fixed finite alphabet.
- A human computer's next action could only depend on the symbol currently being examined and their "state of mind" at that instant. This could result is a new "state of mind".
- Only a finite number of "states of mind" are possible.

These ideas predated the finite automaton and PDA models.

Turing Machine

A *Turing machine* is similar to a *linear bounded automaton* (LBA) except that the tape is "semi-infinite".

- The tape is composed of sequence of cells, where each cell holds one symbol (if no symbol is written to a cell it contains a *blank* symbol).
- The tape has a start cell (the left-most cell). This cell contains the "[" symbol which cannot be overwritten.
- The tape extends infinitely to the right.
- The *read-write head* is centered over one cell at a time and can move one cell to the left or right.

Turing Machine (2)

At each step the Turing machine reads the symbol under the read-write head, replaces the by another symbol (could be the same symbol) and then performs one of three possible actions $\mathcal{A} \in \{L, R, S\}$, where:

- L denotes "Left", move the read-write head one cell to the left.
- R denotes "Right", move the read-write head one cell to the right.
- S denotes "Stationary", read-write head does not move.

Because the tape is "semi-infinite" this introduces a new issue to be considered. The Turing machine may not halt in either an *accepting state* or a *rejecting state* but continue to process the tape!

Turing Machine (3)

A Turing machine (TM) is a 5-tuple $T = (Q, \Sigma, \Gamma, q_0, \delta)$ where:

- Q is a finite set of states. Two additional states h_a and h_r, the accepting and rejecting states, are not elements of Q.
- Σ is the finite input alphabet.
- Γ is the finite tape alphabet. $\Gamma \supseteq \Sigma$. The *blank* symbol, Δ is not an element of Γ.
- $q_0 \in Q$ is the initial state.
- $\delta : Q \times (\Gamma \cup \{[, \Delta\}) \rightarrow (Q \cup \{h_a, h_r\}) \times (\Gamma \cup \{[, \Delta\}) \times \{R, L, S\}$ is the *transition function*.

If $((q, \sigma), (q', \psi, D)) \in \delta$ then when in state q with σ at the current read-write head position, M will replace σ by ψ, move in direction D and enter state q'. If q' is either h_a or h_r the transition causes T to halt. Executing a L move while the current cell contains "[" will cause T to enter h_r.

Turing Machine (4)

A *configuration* of a Turing machine T is defined as:

$$xqy$$

where q is the current state of T, x are the symbols to the left of read-write head on the tape, the string y is either null or starts under the read-write head and everything after xy on the tape is blank.

A *move* of T is when T changes from one configuration to another.

$$xqy \vdash zrw$$

Turing Machine as a Language Acceptor

A Turing machine T accepts w if

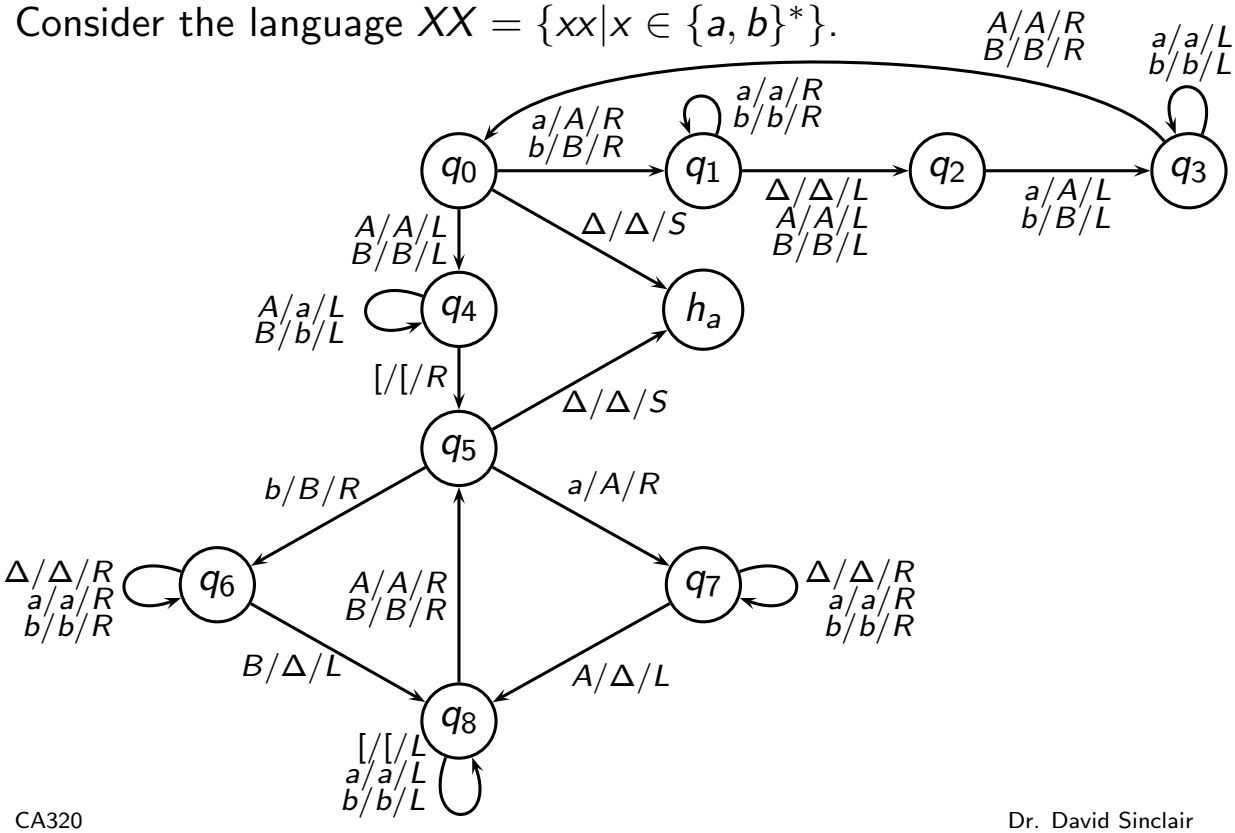
$$[q_0 w \vdash^* x h_a y$$

Let $L = L(T)$ be the language accepted by T. If $x \in L$, T halts in the accepting state h_a. If $x \notin L$, then there are 2 possibilities:

1. T rejects x.
2. T does not halt.

Example 1: TM as a Language Acceptor

Consider the language $XX = \{xx | x \in \{a, b\}^*\}$.



Example 1: TM as a Language Acceptor (2)

We assume that if a transition is not given then the Turing machine moves to the rejecting state h_r.

We can trace the operation of this Turing machine for a few examples.

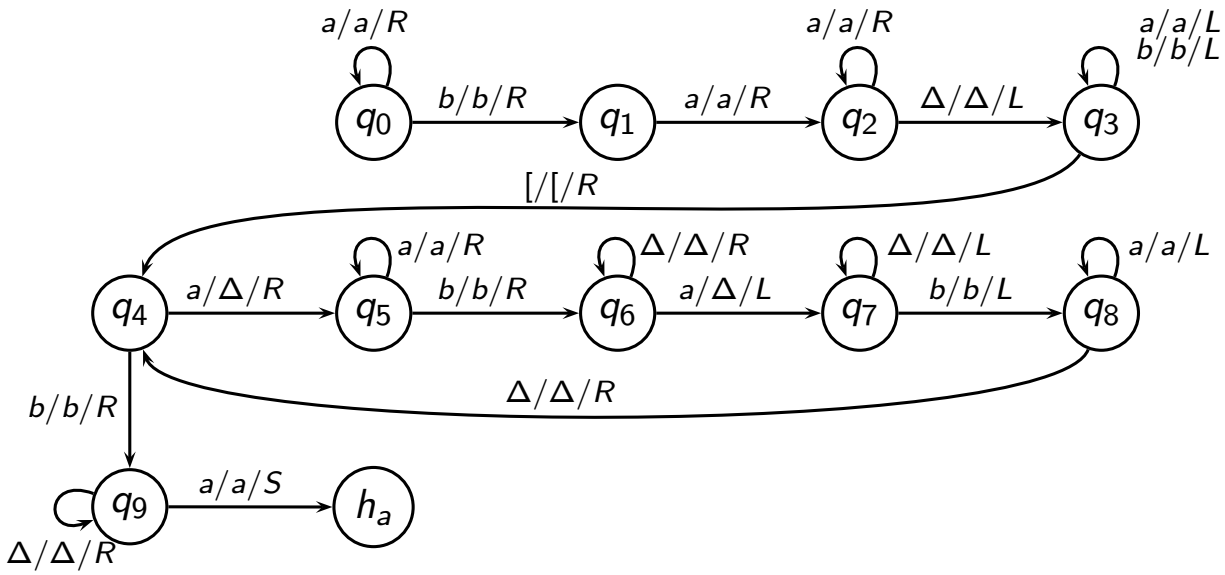
$[q_0aba \vdash [Aq_1ba \vdash^* [Abaq_1\Delta \vdash [Abq_2a \vdash [Aq_3bA$
$\vdash [q_3AbA \vdash [Aq_0bA \vdash [ABq_1A \vdash [Aq_2BA$
$\vdash [Ah_rBA$

$[q_0ab \vdash [Aq_1b \vdash [Abq_1\Delta \vdash [Aq_2b\Delta \vdash [q_3AB$
$\vdash [Aq_0B \vdash [q_4AB \vdash q_4[aB \vdash [q_5aB$
$\vdash [Aq_7B \vdash [Ah_rB$

$[q_0aa \vdash [Aq_1a \vdash [Aaq_1\Delta \vdash [Aq_2a\Delta \vdash [q_3AA$
$\vdash [Aq_0A \vdash [q_4AA \vdash q_4[aA \vdash [q_5aA$
$\vdash [Aq_7A \vdash [q_8A\Delta \vdash [Aq_5\Delta \vdash [Ah_a\Delta$

Example 2: TM as a Language Acceptor

Consider the language $L_{ex2} = \{a^i b a^j | 0 \leq i < j\}$.



Example 2: TM as a Language Acceptor (2)

Consider the input *abaa*

$$
\begin{array}{llll}
[q_0abaq & \vdash [aq_0baa & \vdash [abq_1aa & \vdash [abaq_2a & \vdash [abaaq_2\Delta \\
 & \vdash [abaq_3a & \vdash^* q_3[abaa & \vdash [q_4abaa & \vdash [q_5\Delta baa \\
 & \vdash [\Delta bq_6aa & \vdash [\Delta q_7b\Delta a & \vdash [q_8\Delta b\Delta a & \vdash [\Delta q_4b\Delta a \\
 & \vdash [\Delta bq_9\Delta a & \vdash [\Delta b\Delta q_9a & \vdash [\Delta b\Delta h_aa &
\end{array}
$$

What happens when the input is *aba*? Try this as an exercise.

- Will it halt?
- If it doesn't is that a problem if $aba \notin L_{ex2}$?

TM that Computes Partial Functions

Let $T = (Q, \Sigma, \Gamma, q_0, \delta)$ be a Turing machine, k a natural number and f a partial function on $(\Sigma^*)^k$ with values in Γ^*. We say that T computes f if for every $(x_1, x_2, \ldots, x_k)$ in the domain of f

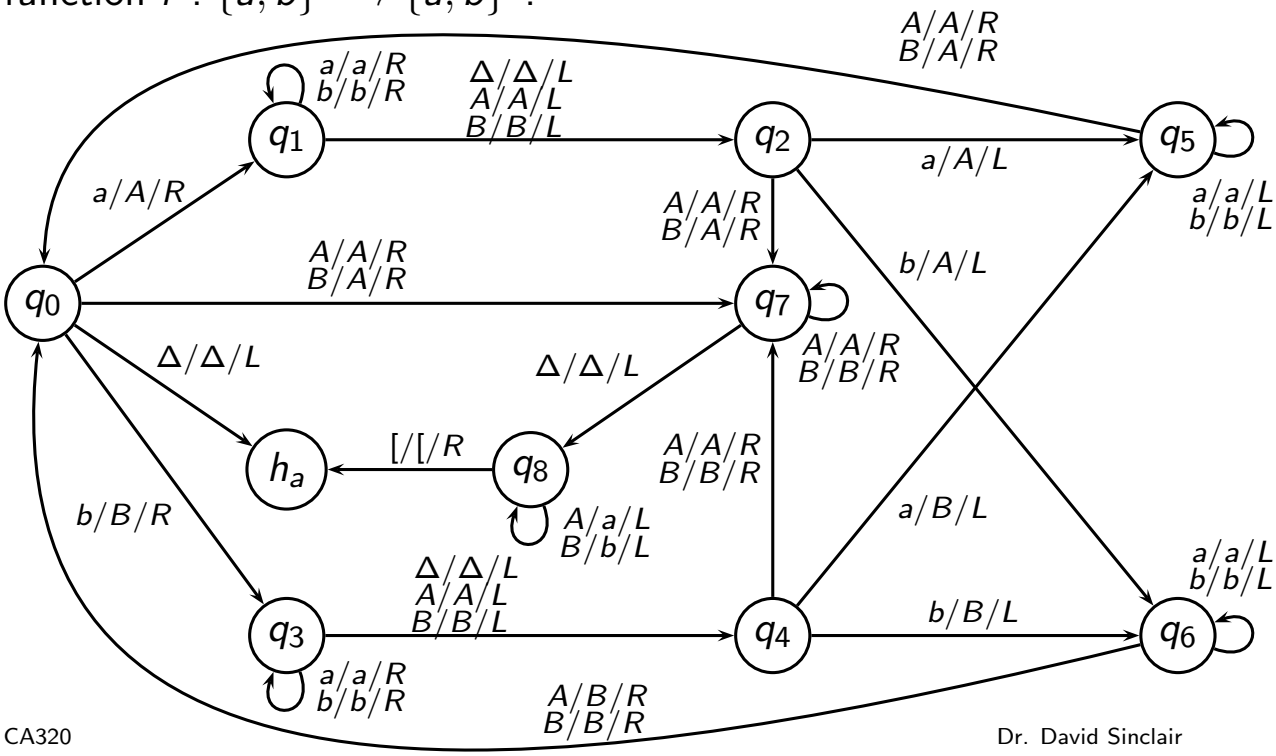
$$[q_0 x_1 \Delta x_2 \Delta \ldots \Delta x_k \vdash^* [h_a f x_1, x_2, \ldots, x_k)$$

and no other input that is a k-tuple of strings is accepted by T.

A partial function $f : (\Sigma^*)^k \to \Gamma^*$ is Turing-computable, or simply computable, if there is a Turing machine that computes f.

Example: TM that Computes Partial Functions

The following diagram is the transition diagram for the reverse function $r : \{a, b\}^* \to \{a, b\}^*$.



Example: TM that Computes Partial Functions (2)

Consider the string *baba*.

$$
\begin{array}{llll}
[q_0baba & \vdash [Bq_3aba & \vdash^* [Babaq_3\Delta & \vdash [Babq_4a & \vdash [Baq_5bB \\
& \vdash^* [q_5BabB & \vdash [Aq_0abB & \vdash [AAq_1bB & \vdash [AAbq_1B \\
& \vdash [AAq_2bB & \vdash [Aq_6AAB & \vdash [ABq_0AB & \vdash [ABAq_7B \\
& \vdash [ABABq_7\Delta & \vdash [ABAq_8B & \vdash^* q_8[abab & \vdash [h_aabab
\end{array}
$$

Functions on Natural Numbers

Represent numbers in unary notation using only the symbol 1 (zero is represented by the empty string).

$f : \mathcal{N} \rightarrow \mathcal{N}$ is computed by M if M computes $f' : \{1\}^* \rightarrow \{1\}^*$ where $f'(1^n) = 1^{f(n)}$, $\forall n \in \mathcal{N}$.

Example: $f(n) = n + 1$, $\forall n \in \mathcal{N}$.

State	Symbol	$\delta(State, Symbol)$
q_0	1	$(q_0, 1, R)$
q_0	Δ	$(q_1, 1, L)$
q_1	1	$(q_1, 1, L)$
q_1	[	$(h_a, [, R)$

$$[q_011 \vdash [1q_01 \vdash [11q_0\Delta \vdash [1q_111 \vdash [q_1111 \vdash [h_a111$$

Functions on Natural Numbers (2)

The following Turing machine computes the addition of 2 unary numbers, m and n, where the initial configuration is $[q_0 m \Delta n$.

State	Symbol	$\delta(State, Symbol, Action)$
q_0	1	$(q_0, 1, R)$
q_0	Δ	(q_1, Δ, R)
q_1	1	$(q_1, 1, R)$
q_1	Δ	(q_2, Δ, L)
q_2	1	(q_3, Δ, L)
q_2	Δ	(q_3, Δ, L)
q_3	1	$(q_3, 1, L)$
q_3	Δ	$(q_3, 1, L)$
q_3	[	$(h_a, [, R)$

Recursive Languages

A language $L \subseteq \Sigma^*$ is *recursive* (also called *Turing-decidable*) iff the characteristic function $\chi_L : \Sigma^* \rightarrow 0, 1$ is Turing-computable, where $\forall w \in \Sigma^*$,

$$\chi_L(w) = \begin{cases} 1, & \text{if } w \in L \\ 0, & \text{otherwise} \end{cases}$$

For example, let $\Sigma = \{a\}$ and $L = \{w \in \Sigma^* | \ |w| \text{ is even}\}$.

The Turing machine that calculates the characteristic function χ_L scans w symbols by symbol from left to right and uses 2 states to determine if the number of symbols it has scanned so far is odd or even. When a blank is reached, it uses its states to remember if the number of symbols was odd or even, moves from right to left, overwriting the contents of the tape and once the start of tape symbol has been found writes a '1' or '0' to the tape.

Recursively Enumerable Languages

A Turing machine M accepts a string w if M halts on the input w.

M accepts a language L iff M halts on w iff $w \in L$.

A language is *recursively enumerable* (also called *Turing-acceptable* or *semi-decidable*) if there is some Turing machine that accepts it.

For example, let $\Sigma = \{a, b\}$ and $L = \{w \in \Sigma^* | \ w$ contains at least one $a\}$.

State	Symbol	$\delta(State, Symbol)$
q_0	a	(h_a, a, S)
q_0	b	(q_0, b, R)

Every recursive language is recursively enumerable.

Combining Turing Machines

Two Turing machine computations M_1 and M_2 can be combined into a larger machine:

- M_1 prepares string as input to M_2.
- M_1 passes control to M_2 with the read-write head at the start of the input.
- M_1 retrieves control when M_2 has completed its computation.

Some basic Turing machines are:

- *Symbol-writing machine* M_a for each symbol $a \in \Sigma$.
- *Head-moving machines* R and L that move the read-write head right and left respectively.

Combining Turing Machines (2)

These basic machines can be combined to perform more complex operations:

- If the current symbol $a = b$, then do M_1 else do M_2.
  IF $(a = b,\ M_1,\ M_2)$
- Move read-write head right until a blank is found.
  R_Δ
- Move read-write head left until a blank is found.
  L_Δ
- Copy machine, transform $w\Delta$ to $w\Delta w\Delta$.
  C = IF $(a = \Delta, R_\Delta, M_\Delta R_\Delta R_\Delta M_a L_\Delta L_\Delta M_a RC)$
- Shift machine, transform $\Delta w\Delta$ to $w\Delta$.
  S = IF$(a = \Delta,\ L,\ M_\Delta L M_a RRS)$

Extensions

The following extensions **do not** increase the power of Turing machines.

- 2-way infinite tape
- Multiple tapes
- Multiple read-write heads on one tape
- 2-dimensional tape
- Nondeterminism

Computable Functions

In order to characterise those functions from $\mathcal{N}$ to $\mathcal{N}$ that can be computed we need to define some *basic* functions.

- $\text{zero}_k(n_1, \ldots, n_k) = 0$
- $\text{succ}(n) = n + 1,\ \forall n \in \mathcal{N}$
- $p_i(n_1, \ldots, n_k) = n_i\ \forall 1 \leq i \leq n$

and a way of *composing* existing functions to define new functions:

$$f(x) = h(g_1(x), \ldots, g_m(x))$$

Primitive Recursion

We can construct a *primitive recursive* function f from existing functions h and g:

$$\begin{aligned} f(x, 0) &= h(x) \\ f(x, \text{succ}(y)) &= g(x, y, f(x, y)) \end{aligned}$$

For example,

$$\begin{aligned} \text{plus}(m, 0) &= m \\ \text{plus}(m, \text{succ}(n)) &= \text{succ}(\text{plus}(m, n)) \end{aligned}$$

$$\begin{aligned} \text{mult}(m, 0) &= \text{zero}(m) \\ \text{mult}(m, \text{succ}(n)) &= \text{plus}(m, \text{mult}(m, n)) \end{aligned}$$

All primitive recursive functions are *total* recursive functions.

Primitive Recursion (2)

Not all computable functions from $\mathcal{N}$ to $\mathcal{N}$ are primitive recursive. The Ackerman function is an example of a computable function that is not primitive recursive.

$$\begin{aligned} \text{Ack}(0, n) &= \text{succ}(n) \\ \text{Ack}(\text{succ}(m), 0) &= \text{Ack}(m, 1) \\ \text{Ack}(\text{succ}(m), \text{succ}(n)) &= \text{Ack}(m, \text{Ack}(\text{succ}(m), n)) \end{aligned}$$

μ-Recursion

For a predicate P, the *unbounded minimisation of* P is the function f defined as follows.

$$f(x) = \min\{y | P(x, y) \text{ is true}\}$$

i.e. the least value of y that satisfies $P(x, y)$.
This is denoted as:

$$f(x) = \mu y[P(x, y)]$$

functions defined in this way are called μ-*recursive* functions.
For example:

$$\text{minus}(x, y) = \mu z[y + z = x]$$

All partial recursive functions are μ-recursive functions.

Theorem

A function is μ-recursive iff it is computable.

Gödelisation

We have seen how we can convert numbers to strings by using unary notation.

We can convert strings to unique numbers by assigning each character in Σ a unique number. The i^{th} character is assigned the value of the i^{th} prime number.

Denoting the i^{th} prime number as p_i, then the string $S = s_1, s_2, \ldots, s_k$ is represented as:

$$g(s_1, s_2, \ldots, s_k) = p_1^{i_1} p_2^{i_2} \ldots p_k^{i_k}$$

So CAT is $2^3 3^1 5^{20}$.

We can use μ-recursive functions to compute functions from strings to strings.

Universal Turing Machine

All the Turing machines we have looked at so far have been specific to a given algorithm. Each new algorithm requires a new Turing machine. Turing in his 1936 paper also describe the *Universal Turing machine* that could execute any algorithm provided it received a description of the algorithm and the data the algorithm is to operate on. The description of the algorithm is in terms of the Turing machine that implements the algorithm.

A Universal Turing machine T_u receives its input as a string (on the tape). The form of the string is $e(T)e(z)$ where T is an arbitrary Turing machine, z is its input and e is an encoding function whose values are strings in $\{0,1\}^*$. The computation performed by T_u on $e(T)e(z)$ will:

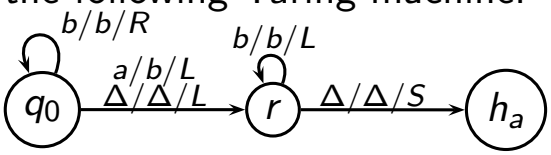
1. accept $e(T)e(z)$ iff T accepts z; and
2. if T accepts z and produces y, then T_u produces $e(y)$.

Universal Turing Machine (2)

Each state, symbol and read-write head direction is assigned a non-zero number:

- $n(h_a) = 1$, $n(h_r) = 2$, $n(q_0) = 3$ and all other states are assigned distinct numbers
- Each symbol a_i, including Δ, is assigned a number $n(a_i)$
- $n(R) = 1$, $n(L) = 2$ and $n(S) = 3$

Consider the following Turing machine:



Universal Turing Machine (3)

We can encode this as follows:

$$n(q_0) = 3, n(r) = 4, n(\Delta) = 1, n(a) = 2 \text{ and } n(b) = 3$$

and then the move $m = \delta(q_0, \Delta) = (r, \Delta, L)$ would be encoded as

$$e(m) = 1^3 0 1 0 1 4 0 1 0 1 2 0 = 1110101111010110$$

and the complete Turing machine T would be

$$\begin{array}{lll} e(T) & = & 1110111011101110 \quad 1110101111010110 \\ & & 11110111011111011110 \quad 111101010101110 \end{array}$$

Church-Turing Thesis

The Church-Turing thesis states that the Turing machine is a general model of computing, that is, that **any** algorithmic procedure that can be carried out, be it by a human computer, a team of human computers or electronic computer, can be carried out by a Turing machine. It was first formulated by Alonzo Church but has never been mathematically proven (mainly because of the difficulty in precisely defining what is meant by an *algorithmic procedure*).

However there is substantial evidence for the thesis, so much evidence that the thesis is generally deemed to be true.

Church-Turing Thesis (2)

Some of the evidence for the Church-Turing thesis is:

1. The nature of the model makes it seem likely that all the crucial step to a computation can be carried out by a Turing machine.
2. All the various enhancements to the Turing machine model, though they may make the model more efficient, have not increased the computational power of the Turing machine.
3. Other theoretical models have been proposed but all have been shown to be equivalent to the Turing machine.
4. Since the conception of the Turing machine model no one has suggested ant type of computation that should be considered as an algorithmic procedure that cannot be implemented by a Turing machine.