Universal Turing Machines

Is the Turing Machine the top of the computational hierarchy? Or, stating it differently, can a Turing Machine simulate ANY Turing Machine? Can you write any Turing Machine computation as the input to a Turing Machine???

We define a class $\mathcal{M}$ of TMs:
1. The states of a DTM $M \in \mathcal{M}$ are $q_1, q_2, \ldots, q_n$, for some $n \geq 1$.
2. The initial state is $q_i$ and the final state is $q_f$.
3. The input alphabet is $\{0, 1\}$.
4. The tape alphabet is $(a_1, a_2, \ldots, a_m)$, $m \geq 3$, with $a_1 = 0$, $a_2 = 1$ and $a_3 = \text{blank}$.

Alphabets: assume we have an alphabet $\Sigma$ with $|\Sigma| = t \geq 1$. Any symbol of $\Sigma$ can be encoded via a string in $\{0, 1\}^*$ of length $\lceil \log_2 t \rceil$.

3/31/08  FCS  1

Universal Turing Machines

We can set up a correspondence $x \in \Sigma^* \leftrightarrow \bar{x} \in (0, 1)^*$, with the string encoding $	ext{proceeding symbol by symbol. Having done this, we can conclude:}$

For every Turing-computable function $f : (\Sigma^*)^n \rightarrow \Sigma^*$ , there exists a DTM $M \in \mathcal{M}$ such that $M$ computes $f : ((0, 1)^*)^n \rightarrow (0, 1)^*$, where $f(\bar{x}_1, \ldots, \bar{x}_n) = \bar{y} \iff f(x_1, \ldots, x_n) = y$.

This allows us to restrict ourselves to the class $\mathcal{M}$, knowing that its machines define the same class of computable functions as the class of all Turing Machines.

We can now try to find a way to encode each of the Turing Machines of $\mathcal{M}$ as the (partial) input to a single Turing Machine, i.e. as a string in $(\{0, 1\}^*)^+$, how? We first need to encode the transition function $\delta(q_i, a_j) = (q_l, a_k, B_p)$.

3/31/08  FCS  2

Universal Turing Machines

3. The starting state is $q_i$.
4. The final state is $q_f$.
5. The blank is $0^\text{blank}$.

Note: the encoding is not 1-to-1. First, any permutation of the instruction sequence will give rise to the same Turing Machine.

Furthermore, many strings in $(\{0, 1\}^*)^+$ will not encode any TM. We can take all of these latter strings and associate them with a machine $M_\text{empty}$ (the empty DTM): $M_\text{empty} = ((q_1, q_2), (a_1, a_2), (a_1, a_2, B_3), (\text{halt state}, \text{halt state}))$, where $\delta = (0, q_1, a_1) = (q_2, q_2, B_3)$, a single instruction that will never be executed: the initial configuration $(q_1, B_3)$ has no successor and the machine halts in a non-halting state without executing any instruction; the only language it accepts is $L(M_\text{empty}) = \emptyset$. We have now set things up so that every string in $(\{0, 1\}^*)^+$ is the code for some TM in $\mathcal{M}$.

Can we recognize the “legal codes”?

3/31/08  FCS  3

Universal Turing Machines

3. The starting state is $q_i$.
4. The final state is $q_f$.
5. The blank is $0^\text{blank}$.

Note: the encoding is not 1-to-1. First, any permutation of the instruction sequence will give rise to the same Turing Machine.

Furthermore, many strings in $(\{0, 1\}^*)^+$ will not encode any TM. We can take all of these latter strings and associate them with a machine $M_\text{empty}$ (the empty DTM): $M_\text{empty} = ((q_1, q_2), (a_1, a_2), (a_1, a_2, B_3), (\text{halt state}, \text{halt state}))$, where $\delta = (0, q_1, a_1) = (q_2, q_2, B_3)$, a single instruction that will never be executed: the initial configuration $(q_1, B_3)$ has no successor and the machine halts in a non-halting state without executing any instruction; the only language it accepts is $L(M_\text{empty}) = \emptyset$. We have now set things up so that every string in $(\{0, 1\}^*)^+$ is the code for some TM in $\mathcal{M}$.

Can we recognize the “legal codes”?

3/31/08  FCS  4

1
Universal Turing Machines

Lemma 5.1. The set \( L = \{ x \in \{0, 1\}^* \mid \exists x \text{ is a legal code string} = x \text{ is of the form } 1^w \cdot \text{code} \ldots \cdot \text{code} \cdot 11, \ k \geq 2 \} \) is primitive recursive.

Proof: let us first identify the characteristics of a legal string \( x \):
1. It is of the form \( 1^w \cdot \text{code} \ldots \cdot \text{code} \cdot 11, \ k \geq 2 \).
2. For each instruction code \( 10^i10^{10}1^j10^k \) of \( x \), \( k \) is either 1 or 2.
3. No two instruction codes start with the same substring \( 10^i10^j \).
4. If the max state is \( q_x \) no instruction begins with \( 10^1 \).

Condition 1:
Claim: the strings of the form \( 1^w \cdot \text{code} \ldots \cdot \text{code} \cdot 11 \) form a regular set.
Proof: each such string has the form \( 12^{10}10^{10}10^{10}1^j11 \), which is a regular expression (regular language).

Claim: the predicate \( \text{instr}(x) = 1 \text{ is an instruction code of } x \)
Proof: by Exercise 4.8.5: or, better yet, 

\[ \begin{align*}
\text{Claim 1:} \quad & \text{Each such string has the form } 1^w \cdot \text{code} \ldots \cdot \text{code} \cdot 11 \quad \text{and primitive recursive.} \\
& \text{The characteristic function of the set defined } \\
& \text{and the primitive recursion of the predicate } \text{or} \text{and of } \text{concat}_n \\
& \text{for all } n \geq 1. \\
\text{Condition 2:} \quad & \text{ } \text{is primitive recursive.} \\
& \text{Thus the characteristic function of the set defined } \\
& \text{and of } \text{concat}_n \\
& \text{for all } n \geq 1. \\
\end{align*} \]

We have already seen that the predicates \( \text{sub}(u, v) = [u \text{ is a substring of } v]\), \( \text{head}(a, v) = [a \text{ is a prefix of } v]\) and \( \text{tail}(a, v) = [a \text{ is a suffix of } v]\) are primitive recursive. We now define the predicate \( \text{instr}(x) = [y \text{ is an instruction code of } x] \)

Since the predicate \( \text{and } \text{is primitive recursive, } \text{sub}(u, v) \text{ is primitive recursive, and } 10^i10^j10^k10^l \) is a regular expression - and thus has a primitive recursive characteristic function, we have that \( \text{instr}(x) \) is primitive recursive. Thus the characteristic function of the set defined by 1. above is primitive recursive.

Condition 2:
\( (\forall y) \text{head}(w) \text{tail}(\text{instr}(x), y) = [k \neq 1 \text{ or } l = 2] \)
Recall that \( (x \Rightarrow y) \equiv (\neg x \text{ or } y) \). Bounded universal quantification of a primitive recursive function is still primitive recursive.

Countability of the set of Turing Computable functions.
The previous discussion allows us to associate with each string \( x \in \{0, 1\}^* \) a Turing Machine \( M_x \in \mathcal{M} \). Furthermore, this machine is the same for all strings \( y \) that encode the code of \( x \). We introduced a lexicographic ordering for strings over an alphabet \( \Sigma \), where \( x_n \) denoted the \( n \)-th string in \( \Sigma \) under this ordering. We will drop \( \Sigma \) when no ambiguity arises. On each such machine, and for each set \( \{0, 1\}^n \), \( k \geq 1 \), we define a function, the partial function computed by the machine: \( M_x = M_x \cup M_x : \{0, 1\}^n \rightarrow \{0, 1\}^* \). Denote the function by \( \phi_x \). It will be defined at \( (x_1, \ldots, x_k) \) with value \( y \in \{0, 1\}^* \), if \( M_x \) halts on input \( (x_1, \ldots, x_k) \) with the final configuration \( (h, a, y) \), undefined otherwise. We extend the output in the case of a final configuration \( (h, a, y) \) to be the longest suffix of \( u \in \{0, 1\}^* \).
Universal Turing Machines

We can now make the statements:
1. The class \( \{ \phi_k(x_1, \ldots, x_k) \mid k \geq 1, n \geq 0 \} \) is the class of partial recursive functions over \( \{0, 1\} \).
2. Every machine \( M_y \) accepts a language
   \[ L(M_y) = \{ x \in \{0, 1\}^* \mid (q_1, x^0) \downarrow_{y, \text{q1}} \} \text{ for some } u \in U, u \in \Gamma^* \]
   \( L(M_y) \) is the domain of \( \phi_y \). If we let \( W_y \) denote \( L(M_y) \), then \( \{ W_y \mid n \geq 0 \} \)
   is exactly the class or recursively enumerable sets over \( \{0, 1\}^* \).

Finally: a) there are only countably many \( M_y \) in \( M \); b) there are only countably many Turing-computable functions.

We will show that there are uncountably many functions on a countable domain (Ex.: \( N \) is the countable domain, \( \chi_y(x) \) for every \( A \subseteq N \) form an uncountable set of functions - to be formally proven later):
1. most functions are non-Turing-computable.

Universal Turing Machines

Proposition 5.2. Let \( \Sigma \) be any finite alphabet.
1. There exists a function \( f : \Sigma \rightarrow \Sigma' \) that is not partial recursive
2. There exists a set \( A \subseteq \Sigma^* \) that is not r.e., and hence not recursive (= not decidable).

We now have a way of encoding all (countably many) possible Turing Machines via our class \( \mathcal{M} \).

Can we now construct a single Turing Machine that will take the encoding of each member of \( \mathcal{M} \) and an appropriate input - both as its input, and simulate that Turing Machine computations on the given input?

After all this work, the answer had better be yes...

Universal Turing Machines

We construct a Universal Turing Machine as a three-tape TM \( U \) over the alphabet \( \{0, 1, B\} \). By previous results this can be converted to a semi-infinite one-tape TM.

Since the machines in \( \mathcal{M} \) can use a bigger alphabet \( \{a_1, a_2, \ldots, a_n\} \), we need to introduce an encoding
\[ x \in \{a_1, a_2, \ldots, a_n\}^* \iff x' \in \{0, 1\}^* \]
\[ a_i = 0^i \cdot x = a_1a_2a_3 \cdots a_n = 0^10^21\cdots 10^n \cdot x' \]
\( U \) uses tape 1 as the (read-only) input tape: the configuration
\( [x', \delta(x')] \) represents a legal code.  The previous proof that such a function is primitive recursive guarantees the computation terminates with a yes/no.  Tape 1 being read-only requires the move (to a read/write tape).

\( y \) is not legal: the machine halts in a state other than \( h \).
\( y \) is legal: \( U \) copies the inputs to tape 2, writes \( 0 \) (= state \( q_0 \)) on tape 3.

Initial configuration:
\[ (s, x', 0^{n+1}, B, B, y) \cup B, 10^110^110^1 \cdots 10^110^110^1, B, y, B) \]

Note that the head of tape 2 is positioned just at the right of the rightmost input, and at the left of the \( 0^* \)-block representing the symbol currently scanned by \( M_y \).  \( B \) is the blank symbol of \( U \), while \( 0^* \) is the blank symbol of \( M_y \).

Universal Turing Machines

Initialization. Copy \( y \) from tape 1 to tape 3. Check that \( y \) represents a legal code.  The previous proof that such a function is primitive recursive guarantees the computation terminates with a yes/no.  Tape 1 being read-only requires the move (to a read/write tape).

\( y \) is not legal: the machine halts in a state other than \( h \).
\( y \) is legal: \( U \) copies the inputs to tape 2, writes \( 0 \) (= state \( q_0 \)) on tape 3.

Initial configuration:
\[ (s, x', 0^{n+1}, B, B, y) \cup B, 10^110^110^1 \cdots 10^110^110^1, B, y, B) \]

Note that the head of tape 2 is positioned just at the right of the rightmost input, and at the left of the \( 0^* \)-block representing the symbol currently scanned by \( M_y \).  \( B \) is the blank symbol of \( U \), while \( 0^* \) is the blank symbol of \( M_y \).
Universal Turing Machines

**Simulation.** $U$ scans $y$ to find an instruction code starting with $10^k10^k$ where $0^k$ is the current content of tape 3 (current state of $M_j$) and $0^k$ is the 0-block to the right of the head of tape 2 (current input symbol for $M_j$).

- If an instruction $10^k10^k10^k10^k10^k$ is found, the $U$ simulates the instruction: It changes the contents of tape 3 to $0^k$, the 0-block to the right of the head of tape 2 to $0^k$ (the symbol written by $M_j$) and moves the head of tape 2 to the next 1 to the right ($h = 0$) or left ($h = 1$). When it moves right, it also checks if this is the rightmost 1 (followed by $B$); if it is, it fills in a new "simulated blank" ($=0^k10^k$). An analogous action takes place on the left: New simulated blank ($=10^{k}$), and move to the 1 on the left.
- If no instruction is found, $U$ checks if the state in tape 3 is $q_0$ (halt state = maxstate(y)). If yes, accept; if no, reject (or enter infinite loop).

---

**Lemma.** The following predicates and functions are primitive recursive:

- $\text{legal}(u, y) = \{u \text{ is a legal code of a configuration of } M_j\}$
- $\text{final}(u, y) = \text{legal}(u, y) \text{ and } y \text{ is a final configuration}$
- $\text{next}(u, u, y) = \{\text{if } \text{final}(u, y) \text{ then } u = v \text{ else } u \leftrightarrow y \}$
- $\text{init}(x_1, \ldots, x_n, y) = \text{the initial configuration of } M_j \text{ on inputs } (x_1, \ldots, x_n)$, encoded as described above.
- $\text{output}(u, y) = \text{the output in } u \text{ if } u \text{ is a final configuration of } M_j \text{ and } = 0 \text{ otherwise}.$
- for each $k \geq 1$, $\text{halt}(x_1, \ldots, x_n, y, t) = \{M_j \text{ halts on inputs } (x_1, \ldots, x_n) \text{ in at most } t \text{ moves}\}$
- for each $k \geq 1$, $\text{print}(x_1, \ldots, x_n, y, t) = \{M_j \text{ halts on inputs } (x_1, \ldots, x_n) \text{ in at most } t \text{ moves and outputs } c\}$.

---

**Theorem 5.3.** For every $k \geq 1$, the partial function $\Phi^k: (\{0, 1\})^* \rightarrow (\{0, 1\})^*$ defined by

$$\Phi^k(x_1, x_2, \ldots, x_n, y) = \Phi^{k_1}(x_1, x_2, \ldots, x_n) \text{ is partial recursive.}$$

**Proof.** Just completed (sketched, really - but tediously completeable).

If we ensure there is no ambiguity between the encoding of states and of input symbols, each configuration of $M_j$

$$(q_0, a_0, a_1, \ldots, a_m, b_0, b_1, \ldots, b_n)$$

can be encoded as $1110^k10^k10^k110^k110^k110^k10^k11$.

The separating pair of 1s identifies the state.

**Def.:** A string $u$ is a legal code of a configuration if it has the (binary) form above.

---

**Proofs.** (some).

- a) $u$, with $u > 5$, and $u' = \text{substr}(u, 3, \text{last}(u))$ is a legal code of a configuration of $M_j$ iff

$$u \in \{0, 1\}^*\{10^k10^k110^k110^k110^k\}^*10^{k+1}$$

- b) $\forall y. \text{next}(u, y) \Rightarrow u' = \text{substr}(u, 3, \text{last}(u))$

- c) $\forall y. \text{next}(u, y) \Rightarrow u' = \text{substr}(u, 3, \text{last}(u))$

where maxstate(y) and maxsym(y) are, respectively, the maximum state and the maximum symbol of $M_j$. Since everything in sight has either been proven to be primitive recursive, or can be easily proven so, the conclusion follows.

**f)** Define a function $f(u, v, y, t)$ to mean that there exists $a_0, a_1, \ldots, a_t$ such that $u = a_0, v = a_t$, and $\text{next}(u_i, y)$ for all $i = 0, 1, \ldots, t-1$. Then $f$ is primitive recursive: proof by "primitive recursion pattern".
Universal Turing Machines

0 steps:
\[ f(u, v, y, 0) = [u = v], \]

\[ t + 1 \text{ steps:} \]
\[ f(u, v, y, t + 1) = (\exists w)_{\text{total}} \left[ \text{next}(u, w, y) \text{ and } f(w, v, y, t) \right]. \]

We now recast the halt predicate:
\[ \text{halt}(x_1, \ldots, x_k, y, t) = (\exists v)_{\text{total}} \left[ \text{next}(u_0, v, y, t) \text{ and } \text{final}(v, y) \right], \]
where \( u_0 = \text{init}(x_1, \ldots, x_k, y) \). Since the latter function is primitive recursive by d), we are done.