Lecture #7: Modelling Computation by Humans — Turing Machines

Lecture Presentation

Considering Another — Much Simpler — Turing Machine Variant

The lecture notes introduced a variant of "Turing machine" that probably resembled Turing machines that you have seen in previous courses, in many respects, but that probably differed because it was designed to compute (partial) functions $f : (\Sigma^\star)^n \rightarrow \Sigma$, for an alphabet Σ and for a positive integer n — so that there might be more than one input string.

At least one older reference — Roger's text, *Theory of Recursive Functions and Effective Computability* [1] — uses a much simpler, and more limited, variant of a Turing machine as the model of computation to be used. This variant is so simple that it might seem surprising that if could be considered to be "equivalent", in computational power, to the other Turing machine variants that you have already seen.

The variant, to be considered, is a version of a "Turing machine" that satisfies the following properties.

- There is a single two-way infinite tape.
- The input alphabet is always $\Sigma = \{\texttt{1}\}$, so that input integers must be given using their unary representations.
- The tape alphabet is always $\Gamma = \{\texttt{1}, \sqcup\}$.
- Only one of the following four instructions can be carried out during a step of the Turing machine
  - `1`: Write `1` onto the cell of the tape that is currently visible, without changing the location of the tape head.
  - $\sqcup$: Write $\sqcup$ onto the cell of the tape that is currently visible, without changing the location of the tape head.

- `L`: Move the tape head one cell to the left without changing the contents of the tape.
- `R`: Move the tape head one cell to the right without changing the contents of the tape.

While there is a designated start state, $q_0 \in Q$ there is no designated "halting" state. Instead, the transition function ***partial*** function

$$\delta : Q \times \{\texttt{1}, \sqcup\} \to Q \times \{\texttt{1}, \sqcup, \texttt{L}, \texttt{R}\}$$

and a computation halts if and only if the machine is in a state $q \in Q$ and a symbol $\sigma \in \{\texttt{1}, \sqcup\}$ is visible on the tape, where $\delta(q, \sigma)$ is undefined.

This kind of ***simple Turing machine*** can be thought of as a 3-tuple

$$\mathcal{M} = (Q, q_0, \delta)$$

where Q is the finite, nonempty set of ***states*** of the Turing machine, $q_0 \in Q$ is the start state, and $\delta : Q \times \{\texttt{1}, \sqcup\} \to Q \times \{\texttt{1}, \sqcup, \texttt{L}, \texttt{R}\}$ is the transition function, as described above.

The initial configuration for this kind of Turing machine is the same as for the kind of Turing machine described in the lecture when $\Sigma = \{\texttt{1}\}$ — so that an integer $k \geq 0$ is represented by its unary representation, that is, a string consisting of k copies of `1`.

Let us use the output requirements used for "weak computation" by Post-Turing machines: If a simple Turing machine, with the above form, halts, then the integer $\ell \geq 0$ that it computes (or "returns as output") is the number ℓ of copies of `1` on the tape when the computation ends.

The goal of this presentation will be to show that the set of computable partial functions $f : \mathbb{N}^n \to \mathbb{N}$ is unchanged, if this is used as the model of computation instead of the more powerful "Turing machine" model presented in the lecture notes — because a simple Turing machine can simulate the computation of a more powerful Turing machine (and vice-versa).

A Reasonably Easy Direction

Claim #1: Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial or total function, for a positive integer k. If there exists a "simple Turing machine" that computes f then f is Turing computable.

Starting the Proof

Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial or total function such that there exists a simple Turing machine

$$\mathcal{M} = (Q, q_0, \delta)$$

that computes f. In order to show that f is Turing-computable we will describe a Turing machine

$$\widehat{\mathcal{M}} = (\widehat{Q}, \Sigma, \Gamma, \widehat{\delta}, \widehat{q}_0, \widehat{q}_{\mathsf{halt}})$$

(as defined in the lecture notes) such that $\widehat{\mathcal{M}}$ also computes f — where it is sufficient to set

$$\Sigma = \{\mathtt{1}\} \qquad \text{and} \qquad \Gamma = \{\mathtt{1}, \#, \sqcup\}.$$

Since $|\Sigma| = 1$ an input $x \in \mathbb{N}$ is represented by the same input string in computations of both $\mathcal{M}$ and $\widehat{\mathcal{M}}$, namely a sting $\mathtt{1}^x$ with length x — the unary representation of the input.

As usual a ***simulation*** will be used to prove the claim — that is, $\widehat{\mathcal{M}}$ will simulate an execution of $\mathcal{M}$ on its input string.

Representing a Configuration

$\widehat{\mathcal{M}}$'s finite control can be used to remember $\widehat{M}$'s state — so we may require that $Q \subseteq \widehat{Q}$. Since $\widehat{\mathcal{M}}$'s finite control will be used to keep track of a finite amount of additional information, as well, the two sets of states, Q and $\widehat{Q}$, will not be the same.

Let $\omega \in \{\sqcup, \mathtt{1}\}^\star$ be the part of $\mathcal{M}$'s tape that is "in use" at any point of $\mathcal{M}$'s computation — so that

- $\mathcal{M}$'s tape stores the string ω with infinitely many copies of "$\sqcup$" (and no copies of $\mathtt{1}$) to the left and to the right of this string;
- $\mathcal{M}$'s tape head points to one of the symbols in ω;
- the leftmost symbol in ω is either $\mathtt{1}$, or is the symbol that $\mathcal{M}$'s tape head points to (or both), and
- the rightmost symbol in ω is either $\mathtt{1}$, or is the symbol that $\mathcal{M}$'s tape head points to (or both).

Then (after initialization, or before the simulation of the next move of $\mathcal{M}$) $\widehat{M}$'s tape stores the string #ω#, with infinitely many copies of "$\sqcup$" to the left and to the right. Thus the additional tape symbol "#" is used to mark the left and right ends of the region of $\mathcal{M}$'s tape that is currently in use. $\widehat{\mathcal{M}}$'s tape head points to the same symbol (in ω) as $\mathcal{M}$'s tape head.

Initialization

Since f is a fixed function the number, k, of natural numbers included in an input is also fixed — and you may describe an "Initialization" phase that depends on (and uses) k.

Why, or How, Does This Matter?

Description of Initialization Phase — and Justification for its Correctness:

Simulating a Move of $\mathcal{M}$

Cases To Be Considered:

-
-
-
-

Details for Cases:

Cleanup

How To Tell When To Enter This Stage:

What Needs To Be Done:

A Few More Details About This (Enough To See That It Can Be Done:

Completing a Proof of the Claim

A More Challenging Direction

Claim #2: Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial or total function, for a positive integer k. If f is Turing-computable then there exists a "simple Turing machine" that computes f.

Making This a Bit Easier

The following claim will make it easier to prove Claim #2.

Claim #3: Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial total function. Let $\Sigma_1 = \{1\}$ and let $\widehat{f} : (\Sigma_1^\star)^k \to \Sigma_1$ such that, for all $x_1, x_2, \ldots, x_k \in \mathbb{N}$, if $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma_1^\star$ such that $\#(\omega_i) = x_i$ (that is, $\omega_i = 1^{x_i}$) for $1 \leq i \leq k$, then

- if $f(x_1, x_2, \ldots, x_k)$ is defined then $\widehat{f}(\omega_1, \omega_2, \ldots, \omega_k)$ is the string $\mu \in \Sigma_1^\star$ such that $\#(\mu) = f(x_1, x_2, \ldots, x_k)$ (that is, $\widehat{f}(1^{x_1}, 1^{x_2}, \ldots, 1^{x_k}) = 1^{f(x_1, x_2, \ldots, x_k)}$), and
- if $f(x_1, x_2, \ldots, x_k)$ is undefined then $\widehat{f}(\omega_1, \omega_2, \ldots, \omega_k)$ is also undefined.

If f is Turing-computable then $\widehat{f}$ is Turing-computable as well.

A proof of Claim #3 will be considered in the exercises for this lecture. It will be assumed to be true, and used, in this lecture presentation.

Starting the Proof

Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial or total function that is Turing-computable — so that, by Claim #3, the corresponding function $\widehat{f} : (\Sigma_1^\star)^k \to \Sigma_1$ (as described in that claim) is Turing-computable as well. Then there exists a Turing machine

$$\mathcal{M} = (Q, \Sigma_1, \Gamma, \delta, q_0, q_{\text{halt}})$$

which computes the function $\widehat{f}$. We will prove Claim #2 by describing a simple Turing machine

$$\widehat{\mathcal{M}} = (\widehat{Q}, \widehat{q}_0, \widehat{\delta})$$

that computes f. Once again, a ***simulation*** is being used to prove this claim — so that $\widehat{\mathcal{M}}$ will simulate an execution of $\mathcal{M}$ on its input string.

Representing a Configuration

Once again, $\widehat{\mathcal{M}}$'s finite control can be used to remember $\widehat{M}$'s state — so we may require that $Q \subseteq \widehat{Q}$. Since $\widehat{\mathcal{M}}$'s finite control will be used to keep track of a finite amount of additional information, as well, the two sets of states, Q and $\widehat{Q}$, will not be the same.

Let $\omega \in \Gamma^\star$ be the part of $\mathcal{M}$'s tape that is "in use" at any point in $\mathcal{M}$'s computation — so that

- $\mathcal{M}$'s tape stores the string ω with infinitely many copies of "blank" (and no copies of any other symbol) to the left and right of this string;
- $\mathcal{M}$'s tape head points to one of the symbols in ω;
- the leftmost symbol in ω is either non-blank, or is the symbol that $\mathcal{M}$'s tape head points to (or both), and
- the rightmost symbol in ω is either non-blank, or is the symbol that $\mathcal{M}$'s tape head points to (or both).

Let $m = |\Gamma|$, so that $m \geq 2$ and

$$\Gamma = \{\sqcup, \mathtt{1}, \sigma_3, \sigma_4, \ldots, \sigma_m\}$$

for "additional" tape symbols $\sigma_3, \sigma_4, \ldots, \sigma_m$. Suppose that we "encode" each symbol $\alpha \in \Gamma$ by a string in $\{\mathtt{1}, \sqcup\}^\star$ with length m:

- $\sqcup$ is encoded by a string, $e(\sqcup)$, consisting of a single copy of "$\mathtt{1}$", followed by $m-1$ copies of "$\sqcup$".
- $\mathtt{1}$ is encoded by a string, $e(\mathtt{1})$, consisting of a pair of copies of "$\mathtt{1}$", followed by $m-2$ copies of "$\sqcup$".
- For $3 \leq i \leq m$, the symbol σ_i is encoded by a string, $e(\sigma_i)$, consisting of i copies of "$\mathtt{1}$", followed by $m-i$ copies of "$\sqcup$".

Now, if $|\omega| = \ell$, so that

$$\omega = \alpha_1 \alpha_2 \ldots \alpha_\ell$$

for symbols $\alpha_1, \alpha_2, \ldots, \alpha_\ell \in \Gamma$, then ω can be represented, on $\widehat{\mathcal{M}}$'s tape, consisting of the following:

- A sequence of $m+1$ copies of "$\mathtt{1}$", followed by a single "$\sqcup$";
- for $1 \leq i \leq \ell$, the string $e(\alpha_i)$ (with length m, as described above), followed by "blank"; and

- another sequence of $m + 1$ copies of "1".

Thus ω is represented, on $\widehat{\mathcal{M}}$'s tape, using a string in $\{1, \sqcup\}^\star$ with length $(\ell + 2) \times (m + 1)$.

If $\mathcal{M}$'s tape head points to the symbol α_i (for $1 \leq i \leq \ell$ then $\widehat{\mathcal{M}}$'s tape head should point to the leftmost symbol in the copy of $e(\alpha_i)$, included in the above string, when the simulation of another step should begin.

Initialization

Since f is a fixed function, and $\widehat{M}$ is a fixed Turing machine, the number, k, of strings (in $\Sigma_1^\star$) included in $\mathcal{M}$'s input, and the size, m, of $\widehat{M}$'s tape alphabet, are both fixed — so that you may describe an "Initialization" phase that depends on (and uses) both k and m.

Why, or How, Does This Matter?

Description of Initialization Phase — and Justification for Its Correctness:

Simulating a Move of $\mathcal{M}$

Cases To Be Considered:

-
-
-

Details for Cases:

Cleanup

How To Tell When To Enter This Stage:

What Needs To Be Done:

A Few More Details About This (Enough To See That It Can Be Done:

Completing a Proof of the Claim

Why Would Anyone Use a "Turing Machine" as Limited as *This*?

References

[1] Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.