Turing machines as function computers

Vassos Hadzilacos

We introduced Turing machines (TMs) as a mathematical model for devices that decide or recognize languages. We can also use TMs for a more general task, namely to compute functions.

Let Σ be an alphabet and consider a function $f\colon \Sigma^* \to \Sigma^*$. How could a TM compute this function? The idea is to start the TM from its initial state with the tape containing any string $x \in \Sigma^*$ and let the TM execute steps until it reaches a special ***halt state*** h, at which point the tape should contain the string $f(x)$, the output of f on input x. Note that the TM now has only one halting state, and that the output is written on the tape. In contrast, the output of TMs that are language deciders is not written on the tape but is reflected in the state when the TM halts: If that state is h_A, the output is interpreted as 1 (or 'yes' or 'true'); if it is h_R, the output is interepted as 0 (or 'no' or 'false'). (So, we can think of TMs that are language deciders as computers of binary functions or solvers of decision problems.)

Formally a TM that computes a function $f\colon \Sigma^* \to \Sigma^*$ is a six-tuple $M = (Q, \Sigma, \Delta, \delta, q_0, h)$, where the first five components are exactly as before, and $h \in Q$ is the halt state.[1] Configurations of the TM M and the "yields" relation $\vdash_M$ are defined exactly as before. We say that M ***computes*** the function $f\colon \Sigma^* \to \Sigma^*$ if, for every $x \in \Sigma^*$, there exist $y_1, y_2 \in \Sigma^*$ such that $q_0 x \vdash_M^* y_1 h y_2$ and $y_1 y_2 = f(x)$. That is, for every string $x \in \Sigma^*$, if we start M in its initial state with x on the tape and the tape head over the leftmost cell, then after a finite number of steps M enters the halt state h and the tape contains the string $f(x)$.[2] A function $f\colon \Sigma^* \to \Sigma^*$ is called ***computable*** or ***recursive*** if there is a TM that computes it. It is important to note that here we require M to halt on every input string x; this is in contrast to the notion of "partial computability", introduced below, where the TM may not halt on some inputs.

We can represent natural numbers as strings in any alphabet Σ. For example, if Σ has a single symbol, say 1, we can represent the natural number n in unary by the string 1^{n+1} (a string of $n+1$ 1s), so that zero is represented by 1, one by 11, two by 111, and so on. If $\Sigma = \{0, 1\}$ we can represent the natural numbers in ordinary binary notation. Thus we can think of any function from the natural numbers to the natural numbers as functions from strings to strings, and we can extend the notion of computable functions (on strings in Σ^*) to functions on natural numbers.

To be concrete, consider $\Sigma = \{0, 1\}$, and for any $x \in \Sigma^*$, let $\mathbf{num}(x)$ be the natural number whose binary representation is x. Then we say that function $F\colon \mathbb{N} \to \mathbb{N}$ is ***computable*** (or ***recursive***) if the function $f\colon \{0,1\}^* \to \{0,1\}^*$ is computable where $f(x)$ is the binary representation of $F\big(\mathbf{num}(x)\big)$. For example the function $F(n) = 2n$ is computable: A TM to compute this function, given the binary representation of n as input, simply scans to the right end of the input and appends a 0 to it (by replacing the leftmost blank on the tape by a 0).

We can further extend the notion of computability to functions on the natural numbers with multiple inputs, such as $\mathbf{Add}(m, n) = m + n$, $\mathbf{Mult}(m, n) = m \cdot n$, etc., in the following way. Let $\Sigma = \{0, 1, \#\}$, where $\#$ is a symbol used to separate binary strings that represent natural numbers. Let $F\colon \mathbb{N}^k \to \mathbb{N}$ be a function on the natural numbers with $k \geq 1$ inputs. Consider a function $f\colon \Sigma^* \to \Sigma^*$, that maps every

[1] There is another minor technical difference: The domain of the transition function is now $(Q - \{h\}) \times \Gamma$ instead of $(Q - \{h_A, h_R\}) \times \Gamma$.

[2] In some definitions the TM is required to halt with the head positioned at the leftmost cell, but we will not require this. The two definitions are essentially equivalent, as it is easy to design the TM to satisfy the additional requirement. In other definitions, there is a designated output tape and $f(x)$ is expected to be written on that tape, starting on the leftmost cell, when the TM halts. Again, this does not change the set of functions that can be computed by TMs.

string of the form $y_1\#y_2\#\ldots\#y_k$, where $y_i \in \{0,1\}^*$, to the binary string that represents the number $F\big(\mathbf{num}(y_1), \mathbf{num}(y_2), \ldots, \mathbf{num}(y_k)\big)$. We say that the function F (on numbers) is computable if the function f (on strings) is computable.

A ***partial function*** from set A to set B, $f\colon A \to B$, is a mathematical concept that is like a function except that it is not defined for certain values of the domain A; so, if f is a partial function from A to B, there is a function from a subset of A to B. Partial functions are important for our study of Turing machines because they are computed by TMs that do not halt on certain inputs, as is the case with some TMs. Examples of partial functions on the natural numbers are the predecessor function $\mathbf{pre}(n)$, which is not defined for $n = 0$, and the logarithm function $\log_2 n$, which is defined (as a function from the natural numbers to the natural numbers) only when n is a power of 2. To emphasize that f is defined on all inputs of its domain (i.e., that f is truly a function) we sometimes say that f is a ***total*** function.

Let $f\colon \Sigma^* \to \Sigma^*$ be a partial function on strings over alphabet Σ. We say that a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, h)$ ***computes the partial function*** f if, (a) for every $x \in \Sigma^*$ such that f is defined for x, $q_0 x \vdash_M^* y_1 h y_2$ for some $y_1, y_2 \in \Sigma^*$ such that $f(x) = y_1 y_2$, and (b) for every $x \in \Sigma^*$ such that f is undefined for x, M loops on x. We say that f is a ***partial computable function*** (or a ***partial recursive function***) if there is a TM M that computes f.

We can extend the notion of partial computable functions on strings to partial computable functions (of one or more inputs) on the natural numbers as we did in the case of total functions.

Turing machines that compute total functions generalize Turing machines that decide languages: Deciding a language is the special case of computing a function with output 1 (indicating that the input string is in the language) or 0 (indicating that the input string is not in the language). Turing machines that compute partial functions generalize Turing machines that recognize languages: An input string that is in the language must result in the TM outputting 1 (accept) but if the string is not in the language the TM may never halt.