mac.1 Variants of Turing Machines

tur:mac:var:sec

There are in fact many possible ways to define Turing machines, of which ours is only one. In some ways, our definition is more liberal than others. We allow arbitrary finite alphabets, a more restricted definition might allow only two tape symbols, 1 and 0. We allow the machine to write a symbol to the tape and move at the same time, other definitions allow either writing or moving. We allow the possibility of writing without moving the tape head, other definitions leave out the N "instruction." In other ways, our definition is more restrictive. We assumed that the tape is infinite in one direction only, other definitions allow the tape to be infinite both to the left and the right. In fact, one can even allow any number of separate tapes, or even an infinite grid of squares. We represent the instruction set of the Turing machine by a transition function; other definitions use a transition relation where the machine has more than one possible instruction in any given situation.

This last relaxation of the definition is particularly interesting. In our definition, when the machine is in state q reading symbol σ, $\delta(q, \sigma)$ determines what the new symbol, state, and tape head position is. But if we allow the instruction set to be a relation between current state-symbol pairs $\langle q, \sigma\rangle$ and new state-symbol-direction triples $\langle q', \sigma', D\rangle$, the action of the Turing machine may not be uniquely determined—the instruction relation may contain both $\langle q, \sigma, q', \sigma', D\rangle$ and $\langle q, \sigma, q'', \sigma'', D'\rangle$. In this case we have a *non-deterministic* Turing machine. These play an important role in computational complexity theory.

There are also different conventions for when a Turing machine halts: we say it halts when the transition function is undefined, other definitions require the machine to be in a special designated halting state. Since the tapes of our turing machines are infinite in one direction only, there are cases where a Turing machine can't properly carry out an instruction: if it reads the leftmost square and is supposed to move left. According to our definition, it just stays put instead, but we could have defined it so that it halts when that happens.

There are also different ways of representing numbers (and hence the input-output function computed by a Turing machine): we use unary representation, but you can also use binary representation. This requires two symbols in addition to 0 and $\triangleright$.

Now here is an interesting fact: none of these variations matters as to which functions are Turing computable. *If a function is Turing computable according to one definition, it is Turing computable according to all of them.*

Photo Credits

Bibliography