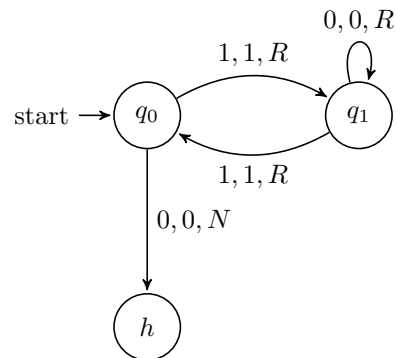


tur.1 Halting States

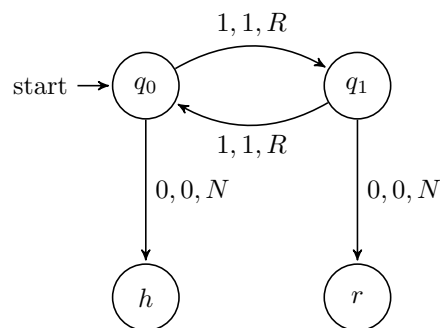
[tur:tur:tur:sec](#) Although we have defined our machines to halt only when there is no instruction [explanation](#) to carry out, common representations of Turing machines have a dedicated *halting state*, h , such that $h \in Q$.

The idea behind a halting state is simple: when the machine has finished operation (it is ready to accept input, or has finished writing the output), it goes into a state h where it halts. Some machines have two halting states, one that accepts input and one that rejects input.

Example tur.1. Halting States. To elucidate this concept, let us begin with an alteration of the even machine. Instead of having the machine halt in state q_0 if the input is even, we can add an instruction to send the machine into a halt state.



Let us further expand the example. When the machine determines that the input is odd, it never halts. We can alter the machine to include a *reject* state by replacing the looping instruction with an instruction to go to a reject state r .



Adding a dedicated halting state can be advantageous in cases like this, [explanation](#) where it makes explicit when the machine accepts/rejects certain inputs. However, it is important to note that no computing power is gained by adding a dedicated halting state. Similarly, a less formal notion of halting has its own advantages. The definition of halting used so far in this chapter makes

the proof of the *Halting Problem* intuitive and easy to demonstrate. For this reason, we continue with our original definition.

Photo Credits

Bibliography