The Halting Problem

Enumerating computable functions

For each $e \in \mathbb{N}$, let $\varphi_e \in \mathbb{N} \rightharpoonup \mathbb{N}$ be the unary partial function computed by the RM with program $prog(e)$. So for all $x, y \in \mathbb{N}$:

$\varphi_e(x) = y$ holds iff the computation of $prog(e)$ started with $\mathtt{R}_0 = 0, \mathtt{R}_1 = x$ and all other registers zeroed eventually halts with $\mathtt{R}_0 = y$.

Thus

$$e \mapsto \varphi_e$$

defines an onto function from $\mathbb{N}$ to the collection of all computable partial functions from $\mathbb{N}$ to $\mathbb{N}$.

An uncomputable function

Let $f \in \mathbb{N} \rightharpoonup \mathbb{N}$ be the partial function with graph $\{(x, 0) \mid \varphi_x(x)\uparrow\}$.

Thus $f(x) = \begin{cases} 0 & \text{if } \varphi_x(x)\uparrow \\ \textit{undefined} & \text{if } \varphi_x(x)\downarrow \end{cases}$

An uncomputable function

Let $f \in \mathbb{N} \rightharpoonup \mathbb{N}$ be the partial function with graph $\{(x, 0) \mid \varphi_x(x)\uparrow\}$.

Thus $f(x) = \begin{cases} 0 & \text{if } \varphi_x(x)\uparrow \\ \textit{undefined} & \text{if } \varphi_x(x)\downarrow \end{cases}$

f is not computable, because if it were, then $f = \varphi_e$ for some $e \in \mathbb{N}$ and hence

- if $\varphi_e(e)\uparrow$, then $f(e) = 0$ (by def. of f); so $\varphi_e(e) = 0$ (since $f = \varphi_e$), hence $\varphi_e(e)\downarrow$
- if $\varphi_e(e)\downarrow$, then $f(e)\downarrow$ (since $f = \varphi_e$); so $\varphi_e(e)\uparrow$ (by def. of f)

—contradiction! So f cannot be computable.

(Un)decidable sets of numbers

Given a subset $S \subseteq \mathbb{N}$, its characteristic function $\chi_S \in \mathbb{N} \to \mathbb{N}$ is given by: $\chi_S(x) \triangleq \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S. \end{cases}$

(Un)decidable sets of numbers

Definition. $S \subseteq \mathbb{N}$ is called (register machine) decidable if its characteristic function $\chi_S \in \mathbb{N} \to \mathbb{N}$ is a register machine computable function. Otherwise it is called undecidable.

So S is decidable iff there is a RM M with the property: for all $x \in \mathbb{N}$, M started with $\mathtt{R}_0 = 0, \mathtt{R}_1 = x$ and all other registers zeroed eventually halts with $\mathtt{R}_0$ containing 1 or 0; and $\mathtt{R}_0 = 1$ on halting iff $x \in S$.

(Un)decidable sets of numbers

Definition. $S \subseteq \mathbb{N}$ is called (register machine) decidable if its characteristic function $\chi_S \in \mathbb{N} \to \mathbb{N}$ is a register machine computable function. Otherwise it is called undecidable.

So S is decidable iff there is a RM M with the property: for all $x \in \mathbb{N}$, M started with $\mathtt{R}_0 = 0, \mathtt{R}_1 = x$ and all other registers zeroed eventually halts with $\mathtt{R}_0$ containing 1 or 0; and $\mathtt{R}_0 = 1$ on halting iff $x \in S$.
Basic strategy: to prove $S \subseteq \mathbb{N}$ undecidable, try to show that decidability of S would imply decidability of the Halting Problem.
For example...

Claim: $S_0 \triangleq \{e \mid \varphi_e(0)\downarrow\}$ is undecidable.

Claim: $S_0 \triangleq \{e \mid \varphi_e(0)\downarrow\}$ is undecidable.

Proof (sketch): Suppose M_0 is a RM computing χ_{S_0}. From M_0's program (using the same techniques as for constructing a universal RM) we can construct a RM H to carry out:

$$
\begin{array}{l}
\mathit{let}\ e = \mathtt{R}_1\ \mathit{and}\ \ulcorner[a_1, \ldots, a_n]\urcorner = \mathtt{R}_2\ \mathit{in} \\
\qquad \mathtt{R}_1 ::= \ulcorner(\mathtt{R}_1 ::= a_1)\,; \cdots ; (\mathtt{R}_n ::= a_n)\,; \mathit{prog}(e)\urcorner\,; \\
\mathtt{R}_2 ::= 0\,; \\
\mathit{run}\ M_0
\end{array}
$$

Then by assumption on M_0, H decides the Halting Problem—contradiction. So no such M_0 exists, i.e. χ_{S_0} is uncomputable, i.e. S_0 is undecidable.

Claim: $S_1 \triangleq \{e \mid \varphi_e \textit{ a total function}\}$ is undecidable.

Claim: $S_1 \triangleq \{e \mid \varphi_e \textit{ a total function}\}$ is undecidable.

Proof (sketch): Suppose M_1 is a RM computing χ_{S_1}. From M_1's program we can construct a RM M_0 to carry out:

$$\begin{array}{l} \textit{let } e = \mathtt{R}_1 \textit{ in } \mathtt{R}_1 ::= \ulcorner \mathtt{R}_1 ::= 0 \,;\, \textit{prog}(e) \urcorner \,; \\ \textit{run } M_1 \end{array}$$

Then by assumption on M_1, M_0 decides membership of S_0 from previous example (i.e. computes χ_{S_0})—contradiction. So no such M_1 exists, i.e. χ_{S_1} is uncomputable, i.e. S_1 is undecidable.

Reductions

If $f : \mathbb{N} \rightarrow \mathbb{N}$ is a *computable* function and $S, T \subseteq \mathbb{N}$ are sets such that:

$$\forall e \in \mathbb{N}. e \in S \text{ if, and only if, } f(e) \in T$$

then if T is decidable, then so is S.

Reductions

If $f : \mathbb{N} \to \mathbb{N}$ is a *computable* function and $S, T \subseteq \mathbb{N}$ are sets such that:

$$\forall e \in \mathbb{N}. e \in S \text{ if, and only if, } f(e) \in T$$

then if T is decidable, then so is S.

For

$$S_0 = \{e \in \mathbb{N} \mid \phi_e(0) \downarrow\} \quad S_1 = \{e \in \mathbb{N} \mid \forall x \in \mathbb{N}. \phi_e(x) \downarrow\}$$

we defined a function f such that $f(e) \in S_1$ if, and only if, $e \in S_0$.

Reductions

If $f : \mathbb{N} \to \mathbb{N}$ is a *computable* function and $S, T \subseteq \mathbb{N}$ are sets such that:

$$\forall e \in \mathbb{N}. e \in S \text{ if, and only if, } f(e) \in T$$

then if T is decidable, then so is S.

For

$$S_0 = \{e \in \mathbb{N} \mid \phi_e(0) \downarrow\} \quad S_1 = \{e \in \mathbb{N} \mid \forall x \in \mathbb{N}. \phi_e(x) \downarrow\}$$

we defined a function f such that $f(e) \in S_1$ if, and only if, $e \in S_0$.

We can then deduce the *un*decidability of S_1 from the *un*decidability of S_0

Reductions

If $f : \mathbb{N} \rightarrow \mathbb{N}$ is a *computable* function and $S, T \subseteq \mathbb{N}$ are sets such that:

$$\forall e \in \mathbb{N}. e \in S \text{ if, and only if, } f(e) \in T$$

then if T is decidable, then so is S.

For

$$S_0 = \{e \in \mathbb{N} \mid \phi_e(0) \downarrow\} \quad S_1 = \{e \in \mathbb{N} \mid \forall x \in \mathbb{N}. \phi_e(x) \downarrow\}$$

we defined a function f such that $f(e) \in S_1$ if, and only if, $e \in S_0$.

The function f is such that $\forall e, x. \phi_{f(e)}(x) \equiv \phi_e(0)$.

Reductions

If $f : \mathbb{N} \to \mathbb{N}$ is a *computable* function and $S, T \subseteq \mathbb{N}$ are sets such that:

$$\forall e \in \mathbb{N}. e \in S \text{ if, and only if, } f(e) \in T$$

then if T is decidable, then so is S.

For

$$S_0 = \{e \in \mathbb{N} \mid \phi_e(0) \downarrow\}$$

"Kleene equivalence": either both sides are undefined or both are defined and equal

we defined a function f such that $f(e) \in S_1$ if, and only if, $e \in S_0$.

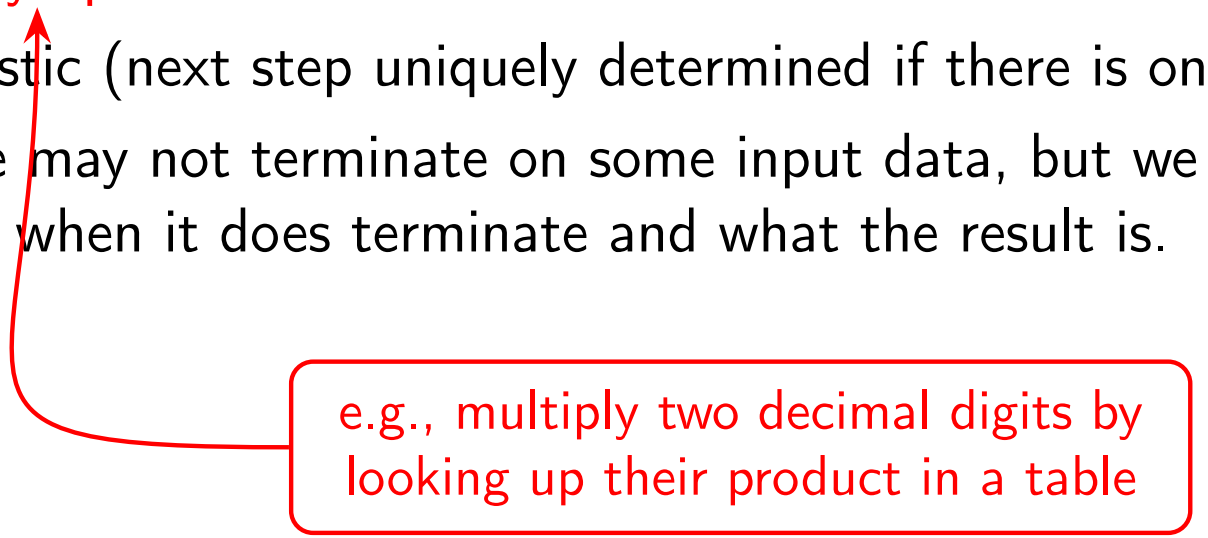
The function f is such that $\forall e, x. \phi_{f(e)}(x) \equiv \phi_e(0)$.

Turing machines

Algorithms, informally

No precise definition of "algorithm" at the time Hilbert posed the *Entscheidungsproblem*, just examples.
Common features of the examples:

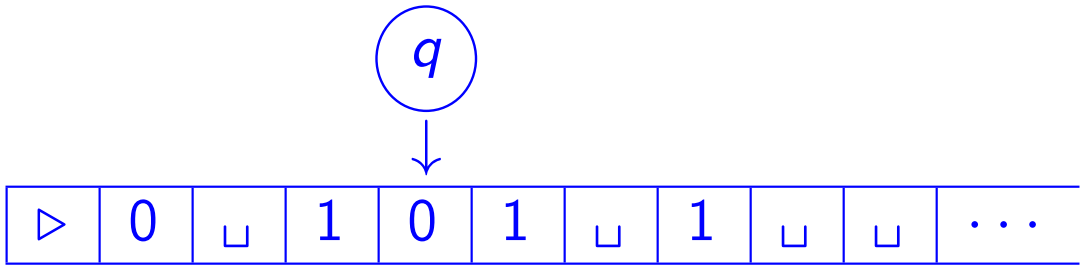
- finite description of the procedure in terms of elementary operations
- deterministic (next step uniquely determined if there is one)
- procedure may not terminate on some input data, but we can recognize when it does terminate and what the result is.



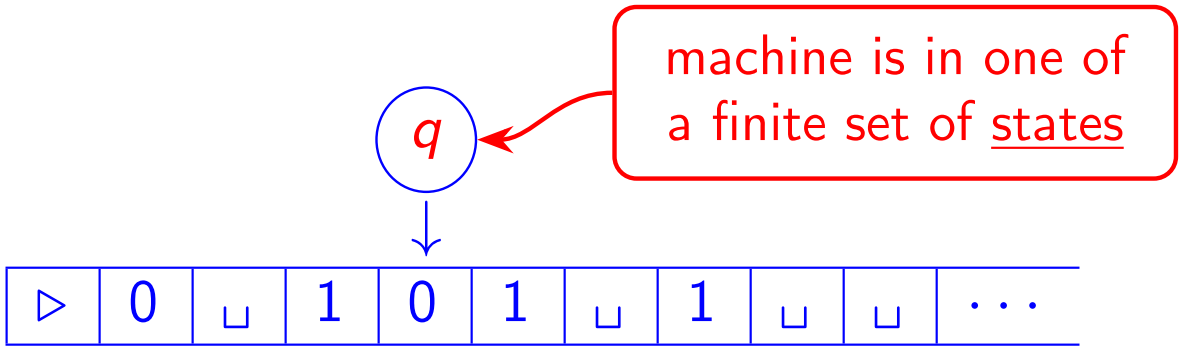
Register Machine computation abstracts away from any particular, concrete representation of numbers (e.g. as bit strings) and the associated elementary operations of increment/decrement/zero-test.
Turing's original model of computation (now called a Turing machine) is more concrete: even numbers have to be represented in terms of a fixed finite alphabet of symbols and increment/decrement/zero-test programmed in terms of more elementary symbol-manipulating operations.

Register Machine computation abstracts away from any particular, concrete representation of numbers (e.g. as bit strings) and the associated elementary operations of increment/decrement/zero-test.
Turing's original model of computation (now called a Turing machine) is more concrete: even numbers have to be represented in terms of a fixed finite alphabet of symbols and increment/decrement/zero-test programmed in terms of more elementary symbol-manipulating operations.

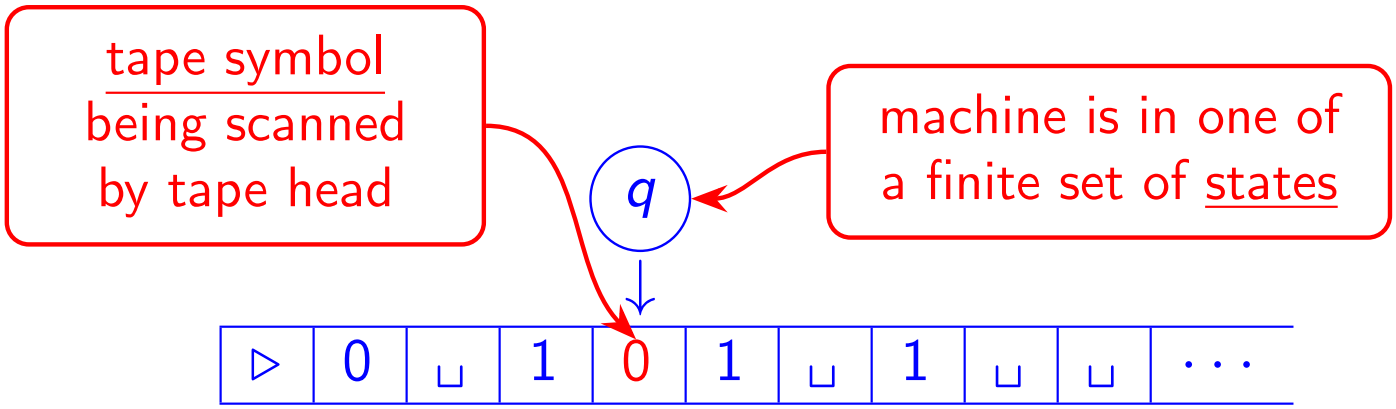
Turing machines, informally



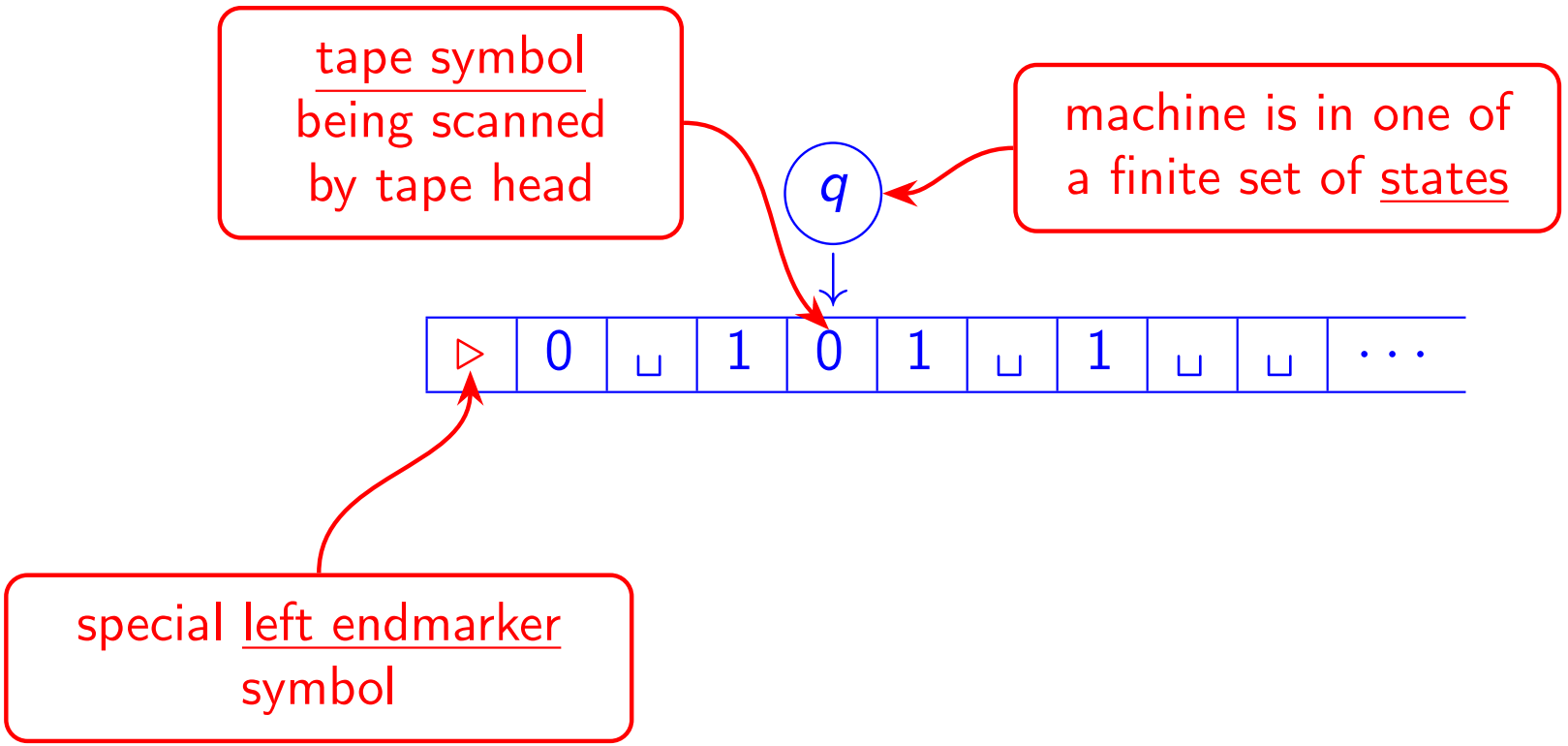
Turing machines, informally



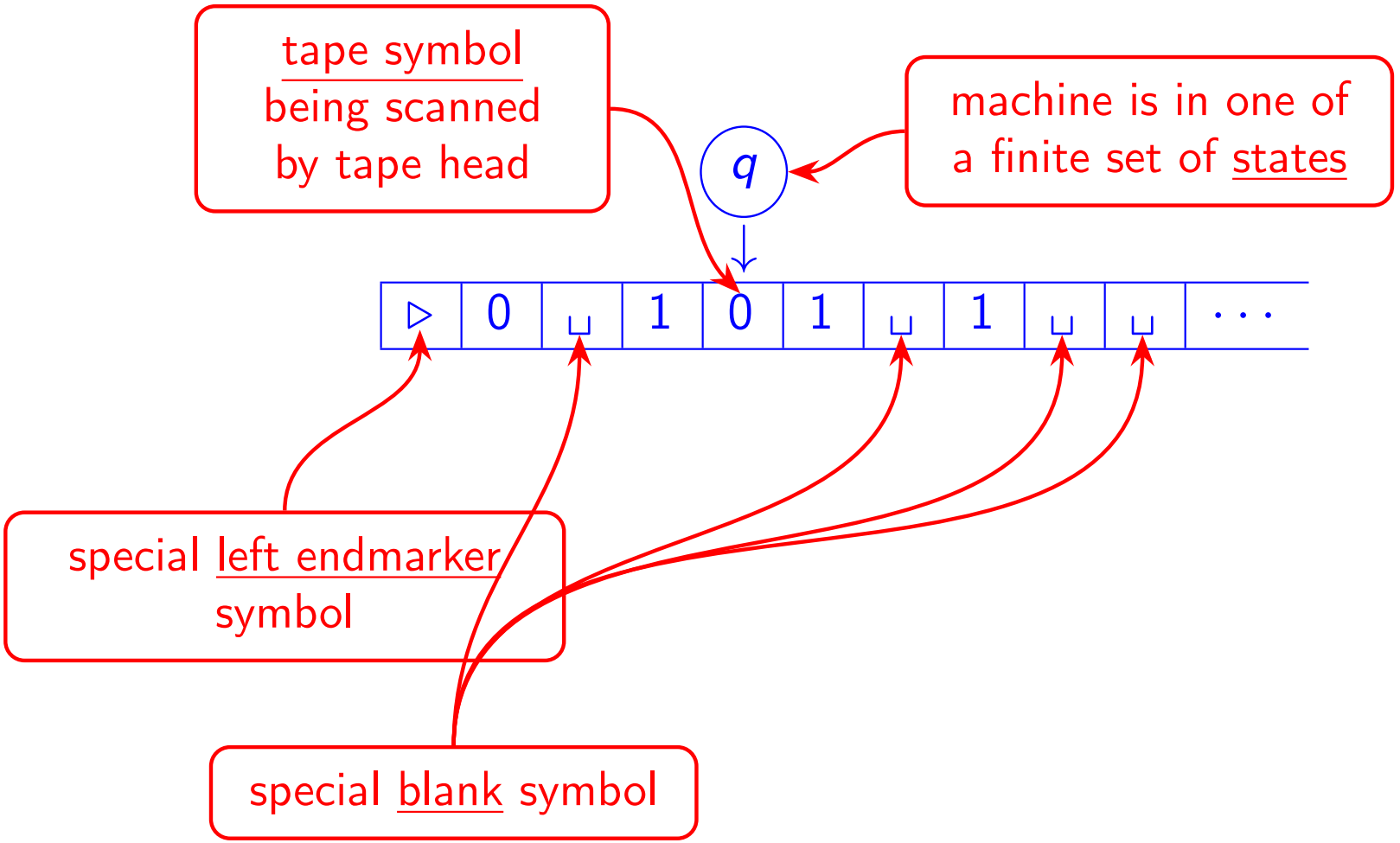
Turing machines, informally



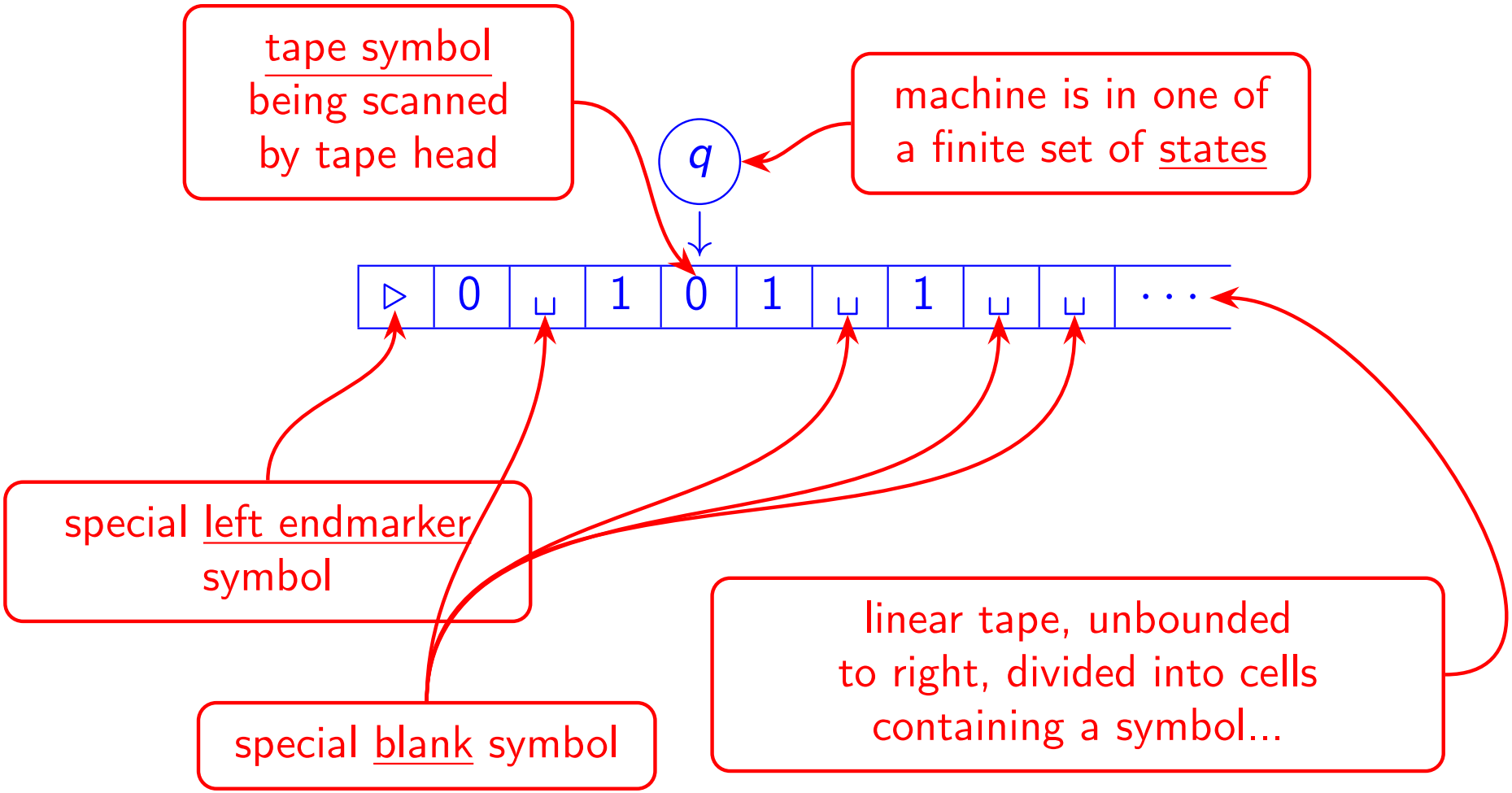
Turing machines, informally



Turing machines, informally



Turing machines, informally



Turing Machines

are specified by:

- Q, finite set of machine states
- Σ, finite set of tape symbols (disjoint from Q) containing distinguished symbols $\triangleright$ (left endmarker) and $\sqcup$ (blank)
- $s \in Q$, an initial state
- $\delta \in (Q \times \Sigma) \rightarrow (Q \cup \{\texttt{acc}, \texttt{rej}\}) \times \Sigma \times \{L, R, S\}$, a transition function—specifies for each state and symbol a next state (or accept `acc` or reject `rej`), a symbol to overwrite the current symbol, and a direction for the tape head to move (L=left, R=right, S=stationary).

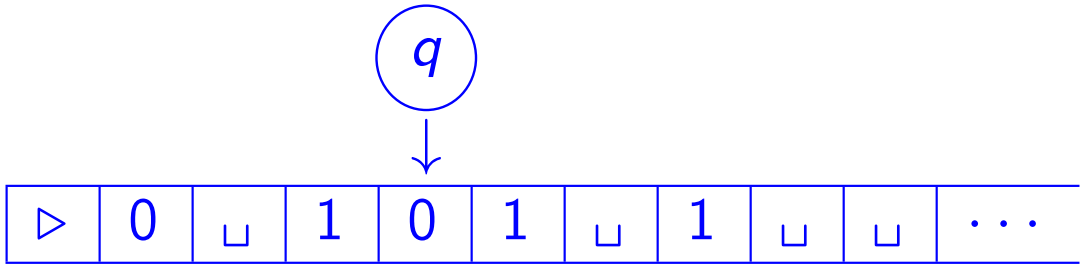
Turing Machines

are specified by:

- Q, finite set of machine states
- Σ, finite set of tape symbols (disjoint from Q) containing distinguished symbols $\triangleright$ (left endmarker) and $\sqcup$ (blank)
- $s \in Q$, an initial state
- $\delta \in (Q \times \Sigma) \rightarrow (Q \cup \{\texttt{acc}, \texttt{rej}\}) \times \Sigma \times \{L, R, S\}$, a transition function, satisfying:

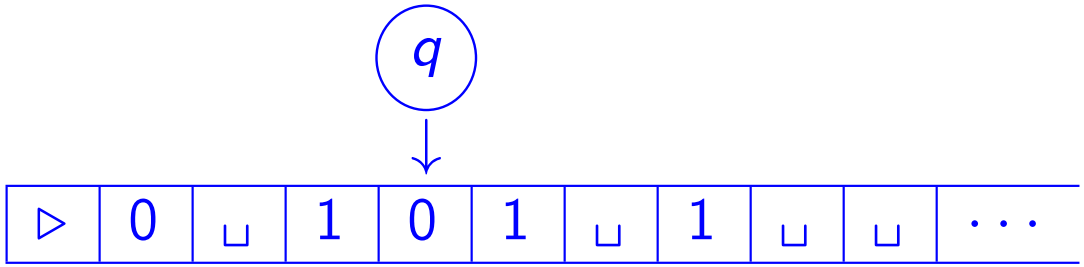
  for all $q \in Q$, *there exists* $q' \in Q \cup \{\texttt{acc}, \texttt{rej}\}$
  with $\delta(q, \triangleright) = (q', \triangleright, R)$
  (i.e. left endmarker is never overwritten and machine always moves to the right when scanning it)

Turing machines, informally



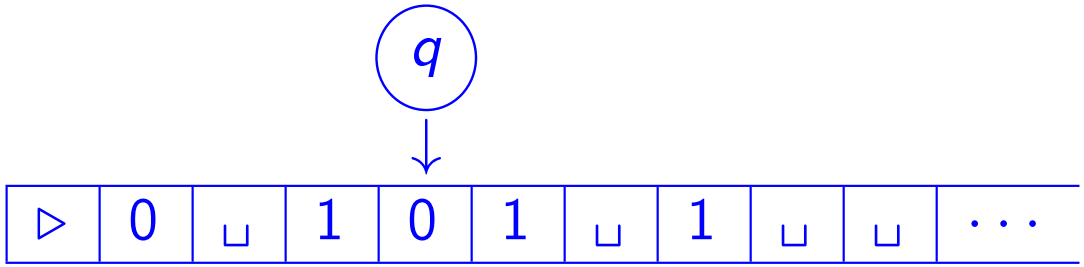
- Machine starts with tape head pointing to the special left endmarker $\triangleright$.

Turing machines, informally



- Machine starts with tape head pointing to the special left endmarker $\triangleright$.
- Machine computes in discrete steps, each of which depends only on current state (q) and symbol being scanned by tape head (0).

Turing machines, informally



- Machine starts with tape head pointing to the special left endmarker $\triangleright$.
- Machine computes in discrete steps, each of which depends only on current state (q) and symbol being scanned by tape head (0).
- Action at each step is to overwrite the current tape cell with a symbol, move left or right one cell, or stay stationary, and change state.