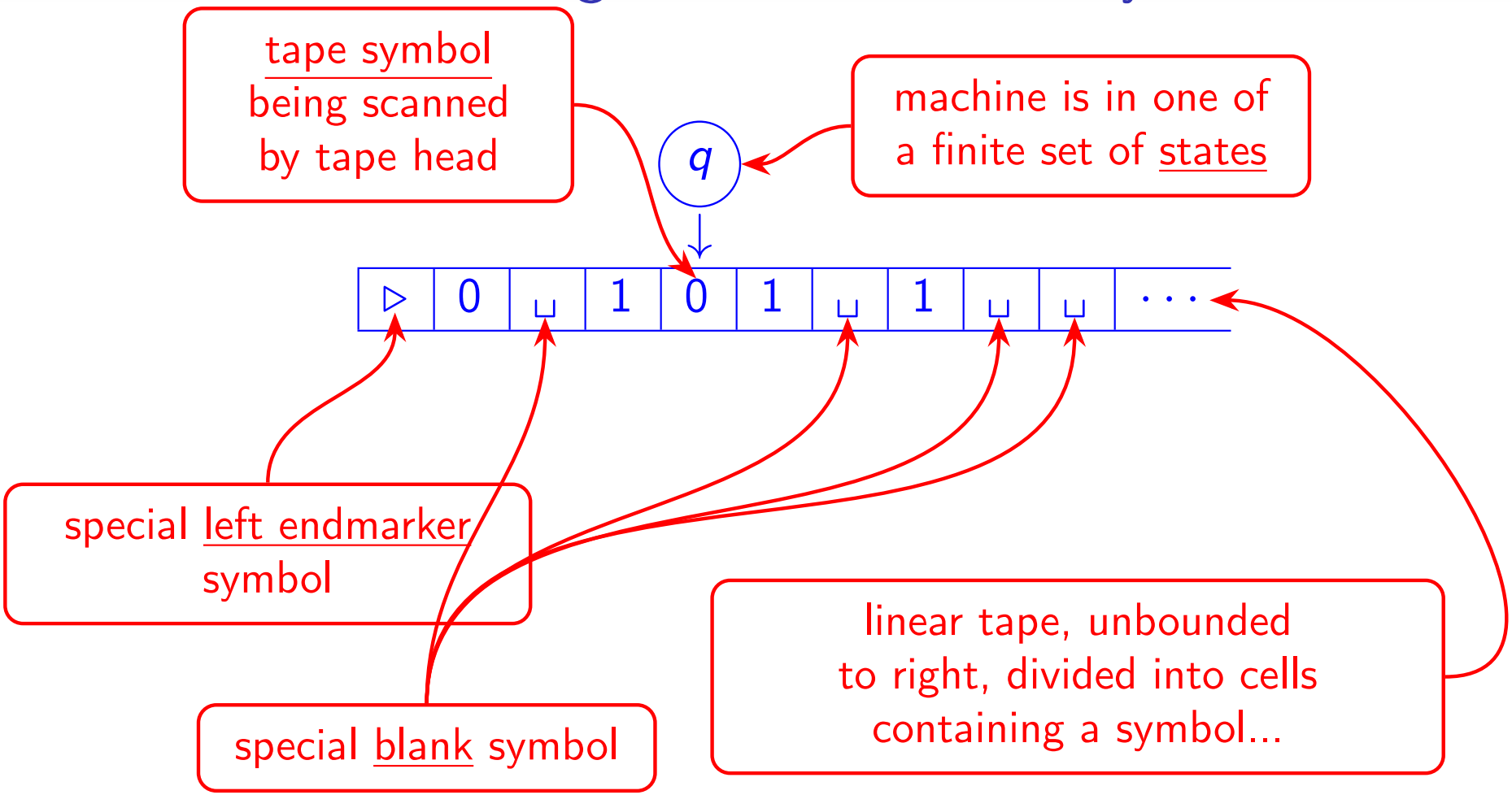
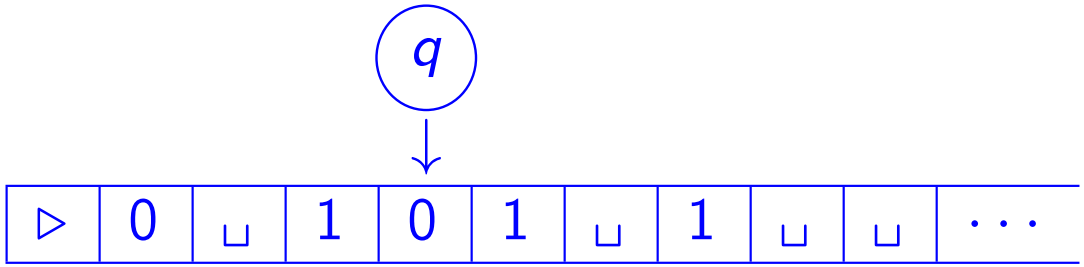
Turing machines

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



- 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.

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)

Example Turing Machine

$M = (Q, \Sigma, s, \delta)$ where
states $Q = \{s, q, q'\}$ (s initial)
symbols $\Sigma = \{\triangleright, \sqcup, 0, 1\}$
transition function
$\delta \in (Q \times \Sigma) \to (Q \cup \{\texttt{acc}, \texttt{rej}\}) \times \Sigma \times \{L, R, S\}$:

δ	$\triangleright$	$\sqcup$	0	1
s	$(s, \triangleright, R)$	$(q, \sqcup, R)$	$(\texttt{rej}, 0, S)$	$(\texttt{rej}, 1, S)$
q	$(\texttt{rej}, \triangleright, R)$	$(q', 0, L)$	$(q, 1, R)$	$(q, 1, R)$
q'	$(\texttt{rej}, \triangleright, R)$	$(\texttt{acc}, \sqcup, S)$	$(\texttt{rej}, 0, S)$	$(q', 1, L)$

Turing machine computation

Turing machine configuration: (q, w, u)
where

- $q \in Q \cup \{\texttt{acc}, \texttt{rej}\}$ = current state
- w = non-empty string ($w = va$) of tape symbols under and to the left of tape head, whose last element (a) is contents of cell under tape head
- u = (possibly empty) string of tape symbols to the right of tape head (up to some point beyond which all symbols are $\sqcup$)

(So $wu \in \Sigma^*$ represents the current tape contents.)

Turing machine computation

Turing machine configuration: (q, w, u)
where

- $q \in Q \cup \{\texttt{acc}, \texttt{rej}\}$ = current state
- w = non-empty string ($w = va$) of tape symbols under and to the left of tape head, whose last element (a) is contents of cell under tape head
- u = (possibly empty) string of tape symbols to the right of tape head (up to some point beyond which all symbols are $\sqcup$)

Initial configurations: $(s, \triangleright, u)$

Turing machine computation

Given a TM $M = (Q, \Sigma, s, \delta)$, we write

$$(q, w, u) \rightarrow_M (q', w', u')$$

to mean $q \neq \texttt{acc}, \texttt{rej}$, $w = va$ (for some v, a) and

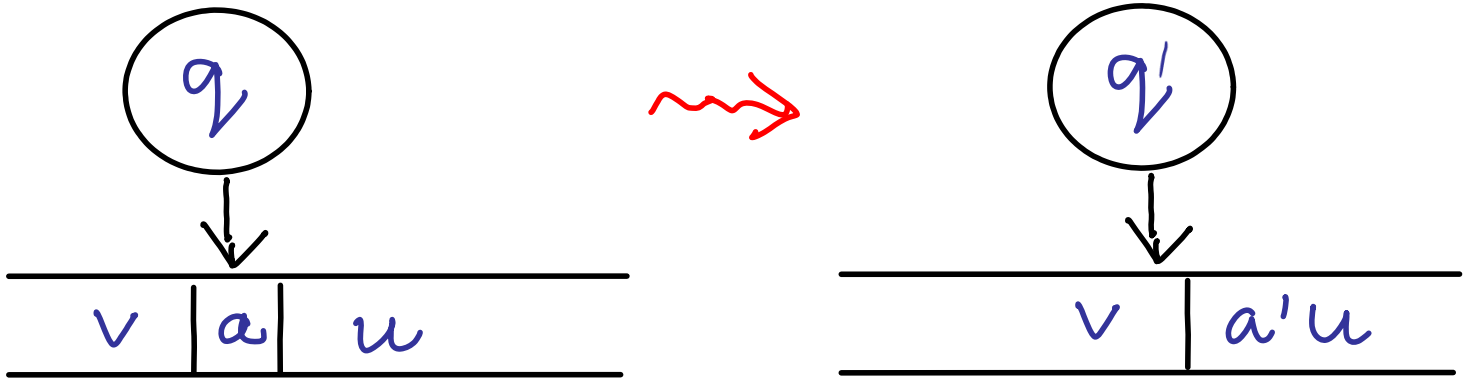
either $\delta(q, a) = (q', a', L)$, $w' = v$, and $u' = a'u$

or $\delta(q, a) = (q', a', S)$, $w' = va'$ and $u' = u$

or $\delta(q, a) = (q', a', R)$, $u = a''u''$ is non-empty, $w' = va'a''$ and $u' = u''$

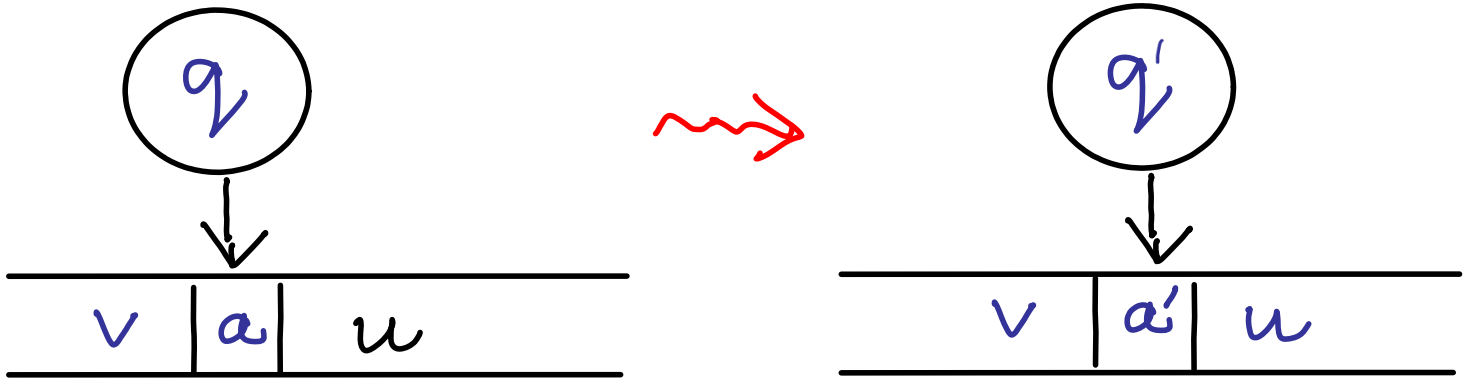
or $\delta(q, a) = (q', a', R)$, $u = \varepsilon$ is empty, $w' = va'\sqcup$ and $u' = \varepsilon$.

$$\delta(q, a) = (q', a', L)$$



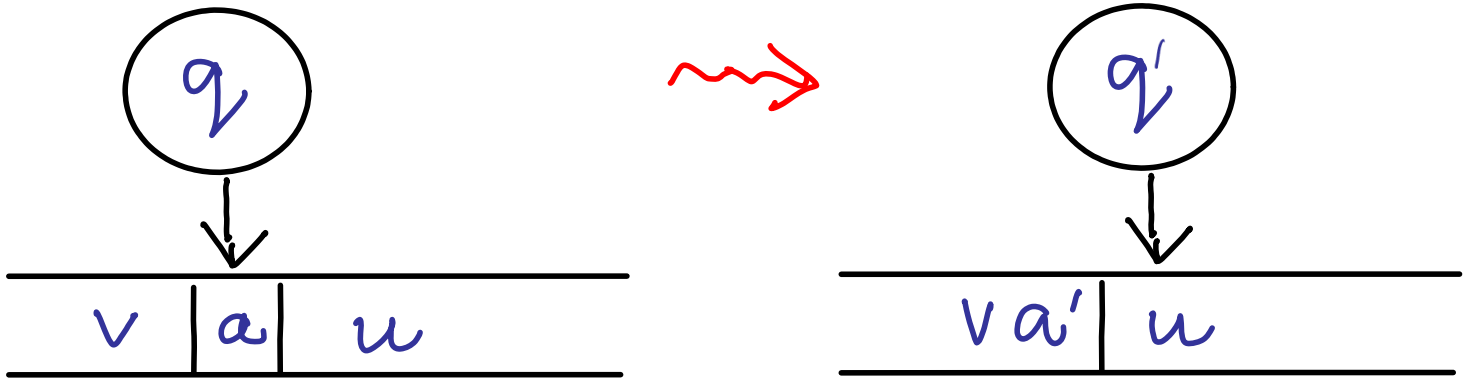
$$(q, va, u) \xrightarrow[M]{} (q', v, a'u)$$

$$\delta(q, a) = (q', a', S)$$



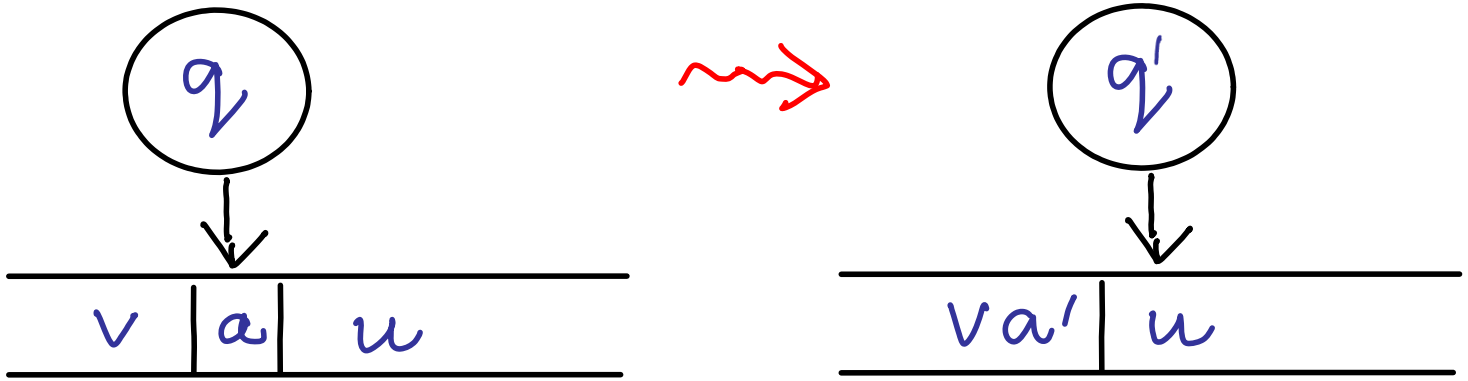
$$(q, va, u) \xrightarrow[M]{} (q', va', u)$$

$$\delta(q, a) = (q', a', R)$$



$$(q, va, u) \xrightarrow{}_{M} (q', ?, ?)$$

$$\delta(q, a) = (q', a', R)$$

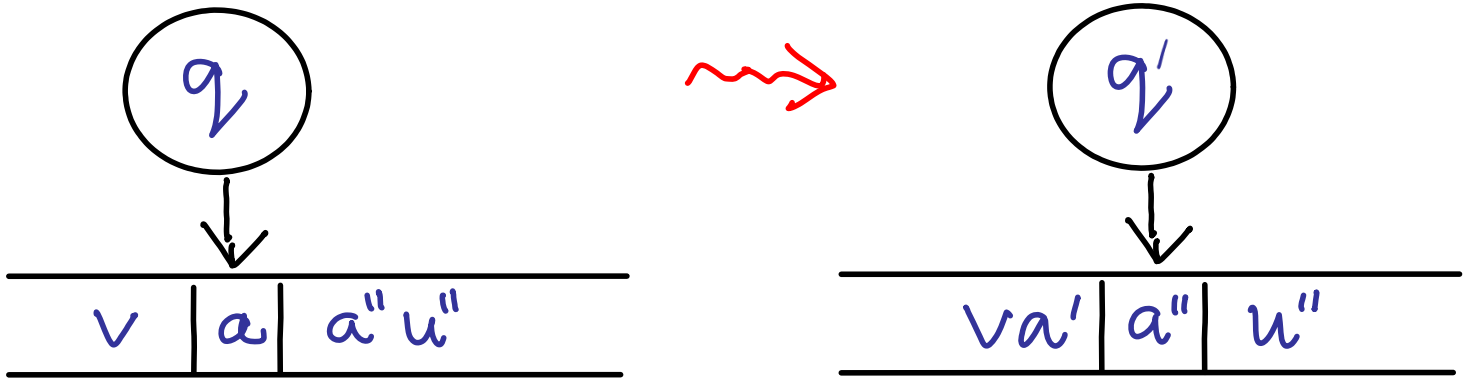


$$(q, va, u) \longrightarrow_M (q', ?, ?)$$

Two cases :

$$\begin{cases} u = a''u'' & \text{is non-empty} \\ u = \varepsilon & \text{is empty} \end{cases}$$

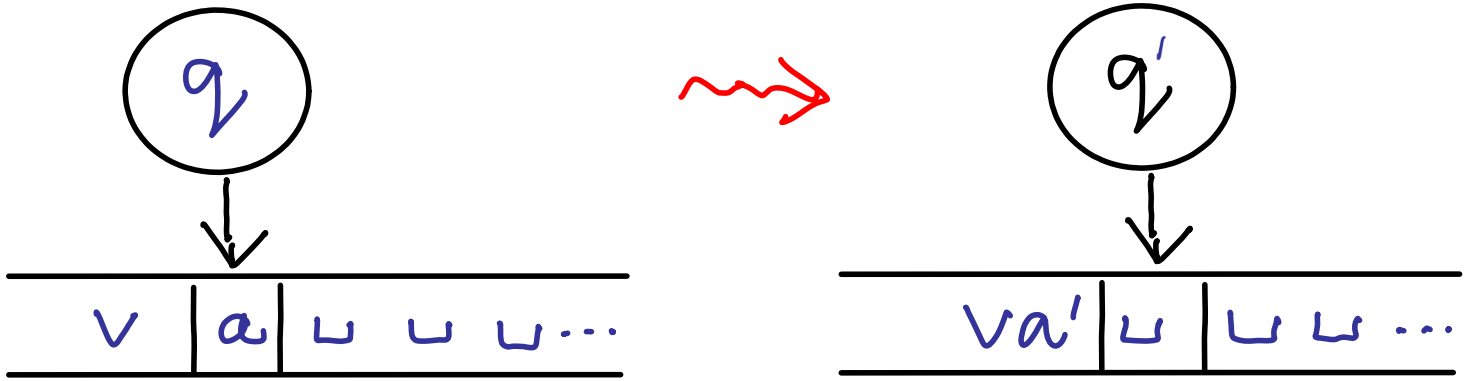
$$\delta(q, a) = (q', a', R)$$



$$(q, va, a''u'') \xrightarrow[M]{} (q', va'a'', u'')$$

Two cases : $\begin{cases} u = a''u'' & \text{is non-empty} \\ u = \varepsilon & \text{is empty} \end{cases}$

$$\delta(q, a) = (q', a', R)$$



$$(q, va, \varepsilon) \xrightarrow[M]{} (q', va'\sqcup, \varepsilon)$$

Two cases : $\begin{cases} u = a''u'' & \text{is non-empty} \\ u = \varepsilon & \text{is empty} \end{cases}$

Turing machine computation

A computation of a TM M is a (finite or infinite) sequence of configurations $c_0, c_1, c_2, \ldots$
where

- $c_0 = (s, \triangleright, u)$ is an initial configuration
- $c_i \rightarrow_M c_{i+1}$ holds for each $i = 0, 1, \ldots$.

The computation

- does not halt if the sequence is infinite
- halts if the sequence is finite and its last element is of the form $(\texttt{acc}, w, u)$ or $(\texttt{rej}, w, u)$.

Example Turing Machine

$M = (Q, \Sigma, s, \delta)$ where
states $Q = \{s, q, q'\}$ (s initial)
symbols $\Sigma = \{\triangleright, \sqcup, 0, 1\}$
transition function
$\delta \in (Q \times \Sigma) \to (Q \cup \{\texttt{acc}, \texttt{rej}\}) \times \Sigma \times \{L, R, S\}$:

δ	$\triangleright$	$\sqcup$	0	1
s	$(s, \triangleright, R)$	$(q, \sqcup, R)$	$(\texttt{rej}, 0, S)$	$(\texttt{rej}, 1, S)$
q	$(\texttt{rej}, \triangleright, R)$	$(q', 0, L)$	$(q, 1, R)$	$(q, 1, R)$
q'	$(\texttt{rej}, \triangleright, R)$	$(\texttt{acc}, \sqcup, S)$	$(\texttt{rej}, 0, S)$	$(q', 1, L)$

Claim: the computation of M starting from configuration $(s, \triangleright, \sqcup 1^n 0)$ halts in configuration $(\texttt{acc}, \triangleright\sqcup, 1^{n+1}0)$.

Example Turing Machine

$M = (Q, \Sigma, s, \delta)$ where
states $Q = \{s, q, q'\}$ (s initial)
symbols $\Sigma = \{\triangleright, \sqcup, 0, 1\}$
transition function
$\delta \in (Q \times \Sigma) \to (Q \cup \{\texttt{acc}, \texttt{rej}\}) \times \Sigma \times \{L, R, S\}$:

δ	$\triangleright$	$\sqcup$	0	1
s	$(s, \triangleright, R)$	$(q, \sqcup, R)$	$(\texttt{rej}, 0, S)$	$(\texttt{rej}, 1, S)$
q	$(\texttt{rej}, \triangleright, R)$	$(q', 0, L)$	$(q, 1, R)$	$(q, 1, R)$
q'	$(\texttt{rej}, \triangleright, R)$	$(\texttt{acc}, \sqcup, S)$	$(\texttt{rej}, 0, S)$	$(q', 1, L)$

Claim: the computation of M starting from configuration $(s, \triangleright, \sqcup 1^n 0)$ halts in configuration $(\texttt{acc}, \triangleright\sqcup, 1^{n+1}0)$.

a string of n 1s

The computation of M starting from configuration $(s\,,\,\triangleright\,,\,\sqcup 1^n 0)$:

$$
\begin{array}{rll}
(s\,,\,\triangleright\,,\,\sqcup 1^n 0) & \rightarrow_M & (s\,,\,\triangleright\sqcup\,,\,1^n 0) \\
& \rightarrow_M & (q\,,\,\triangleright\sqcup 1\,,\,1^{n-1} 0) \\
& \vdots & \\
& \rightarrow_M & (q\,,\,\triangleright\sqcup 1^n\,,\,0) \\
& \rightarrow_M & (q\,,\,\triangleright\sqcup 1^n 0\,,\,\varepsilon) \\
& \rightarrow_M & (q\,,\,\triangleright\sqcup 1^{n+1}\sqcup\,,\,\varepsilon) \\
& \rightarrow_M & (q'\,,\,\triangleright\sqcup 1^{n+1}\,,\,0) \\
& \vdots & \\
& \rightarrow_M & (q'\,,\,\triangleright\sqcup\,,\,1^{n+1} 0) \\
& \rightarrow_M & (\mathtt{acc}\,,\,\triangleright\sqcup\,,\,1^{n+1} 0)
\end{array}
$$

Theorem. The computation of a Turing machine M can be implemented by a register machine.

Proof (sketch).

Step 1: fix a numerical encoding of M's states, tape symbols, tape contents and configurations.

Step 2: implement M's transition function (finite table) using RM instructions on codes.

Step 3: implement a RM program to repeatedly carry out $\rightarrow_M$.

Step 1

- Identify states and tape symbols with particular numbers:

$$\begin{array}{rcl|rcl} \texttt{acc} & = & 0 & \sqcup & = & 0 \\ \texttt{rej} & = & 1 & \triangleright & = & 1 \\ Q & = & \{2, 3, \ldots, n\} & \Sigma & = & \{0, 1, \ldots, m\} \end{array}$$

- Code configurations $c = (q, w, u)$ by:

$$\ulcorner c \urcorner = \ulcorner [q, \ulcorner [a_n, \ldots, a_1] \urcorner, \ulcorner [b_1, \ldots, b_m] \urcorner] \urcorner$$

where $w = a_1 \cdots a_n$ $(n > 0)$ and $u = b_1 \cdots b_m$ $(m \geq 0)$ say.

Step 1

reversal of w makes it easier to use our RM programs for list manipulation

- Code configurations $c = (q, w, u)$ by:

$$\ulcorner c \urcorner = \ulcorner [q, \ulcorner [a_n, \ldots, a_1] \urcorner, \ulcorner [b_1, \ldots, b_m] \urcorner] \urcorner$$

where $w = a_1 \cdots a_n$ $(n > 0)$ and $u = b_1 \cdots b_m$ $(m \geq 0)$ say.

Step 2

Using registers

$\mathtt{Q}$ = current state

$\mathtt{A}$ = current tape symbol

$\mathtt{D}$ = current direction of tape head
(with $L = 0$, $R = 1$ and $S = 2$, say)

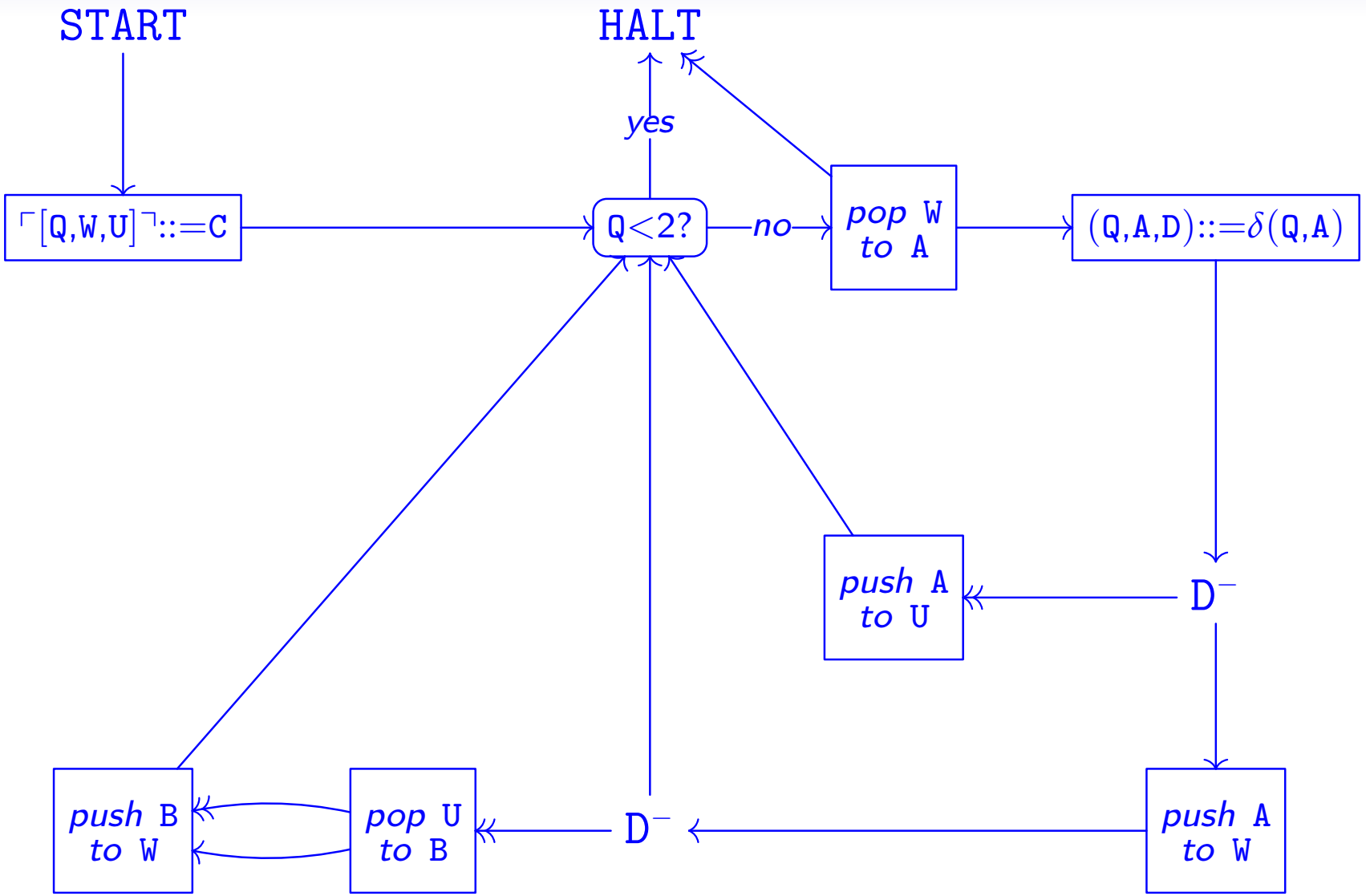
one can turn the finite table of (argument,result)-pairs specifying δ into a RM program $\rightarrow \boxed{(\mathtt{Q}, \mathtt{A}, \mathtt{D}) ::= \delta(\mathtt{Q}, \mathtt{A})} \rightarrow$ so that starting the program with $\mathtt{Q} = q$, $\mathtt{A} = a$, $\mathtt{D} = d$ (and all other registers zeroed), it halts with $\mathtt{Q} = q'$, $\mathtt{A} = a'$, $\mathtt{D} = d'$, where $(q', a', d') = \delta(q, a)$.

Step 3

The next slide specifies a RM to carry out M's computation. It uses registers

- $\mathtt{C}$ = code of current configuration
- $\mathtt{W}$ = code of tape symbols at and left of tape head (reading right-to-left)
- $\mathtt{U}$ = code of tape symbols right of tape head (reading left-to-right)

Starting with $\mathtt{C}$ containing the code of an initial configuration (and all other registers zeroed), the RM program halts if and only if M halts; and in that case $\mathtt{C}$ holds the code of the final configuration.


START
HALT
yes
⌜[Q,W,U]⌝::=C
Q<2?
no
pop W
to A
(Q,A,D)::=δ(Q,A)
push A
to U
D⁻
push B
to W
pop U
to B
D⁻
push A
to W

We've seen that a Turing machine's computation can be implemented by a register machine.
The converse holds: the computation of a register machine can be implemented by a Turing machine.
To make sense of this, we first have to fix a tape representation of RM configurations and hence of numbers and lists of numbers...

Tape encoding of lists of numbers

Definition. A tape over $\Sigma = \{\triangleright, \sqcup, 0, 1\}$ codes a list of numbers if precisely two cells contain 0 and the only cells containing 1 occur between these.

Such tapes look like:

$$\triangleright \sqcup \cdots \sqcup 0 \underbrace{1 \cdots 1}_{n_1} \sqcup \underbrace{1 \cdots 1}_{n_2} \sqcup \cdots \sqcup \underbrace{1 \cdots 1}_{n_k} 0 \underbrace{\sqcup \cdots}_{\text{all } \sqcup\text{'s}}$$

which corresponds to the list $[n_1, n_2, \ldots, n_k]$.

Tape encoding of lists of numbers

Definition. A tape over $\Sigma = \{\triangleright, \sqcup, 0, 1\}$ codes a list of numbers if precisely two cells contain 0 and the only cells containing 1 occur between these.

Such tapes look like:

$$\triangleright \sqcup \cdots \sqcup 0 \underbrace{1 \cdots 1}_{n_1} \sqcup \underbrace{1 \cdots 1}_{n_2} \sqcup \cdots \sqcup \underbrace{1 \cdots 1}_{n_k} 0 \underbrace{\sqcup \cdots}_{\text{all } \sqcup\text{'s}}$$

which corresponds to the list $[n_1, n_2, \ldots, n_k]$.
E.g., $\triangleright 0 \sqcup\sqcup\sqcup 11 \sqcup 1 \sqcup 0 \sqcup \ldots$ codes the list...

Tape encoding of lists of numbers

Definition. A tape over $\Sigma = \{\triangleright, \sqcup, 0, 1\}$ codes a list of numbers if precisely two cells contain 0 and the only cells containing 1 occur between these.

Such tapes look like:

$$\triangleright \sqcup \cdots \sqcup 0 \underbrace{1 \cdots 1}_{n_1} \sqcup \underbrace{1 \cdots 1}_{n_2} \sqcup \cdots \sqcup \underbrace{1 \cdots 1}_{n_k} 0 \underbrace{\sqcup \cdots}_{\text{all } \sqcup\text{'s}}$$

which corresponds to the list $[n_1, n_2, \ldots, n_k]$.
E.g., $\triangleright 0 \sqcup\sqcup\sqcup 11 \sqcup 1 \sqcup 0 \sqcup \ldots$ codes the list... $[0, 0, 0, 2, 1, 0]$

Turing computable function

Definition. $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ is Turing computable if and only if there is a Turing machine M with the following property:

Starting M from its initial state with tape head on the left endmarker of a tape coding $[0, x_1, \ldots, x_n]$, M halts if and only if $f(x_1, \ldots, x_n)\downarrow$, and in that case the final tape codes a list (of length ≥ 1) whose first element is y where $f(x_1, \ldots, x_n) = y$.

Theorem. A partial function is Turing computable if and only if it is register machine computable.

Proof (sketch). We've seen how to implement any TM by a RM. Hence

f TM computable implies f RM computable.

For the converse, one has to implement the computation of a RM in terms of a TM operating on a tape coding RM configurations. To do this, one has to show how to carry out the action of each type of RM instruction on the tape. It should be reasonably clear that this is possible in principle, even if the details (omitted) are tedious.