What 'human computers' can do

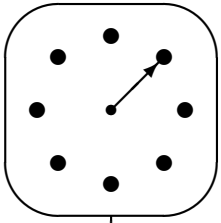
In 1936 **Alan Turing** made an attempt to formulate an abstract model of a **'human computer'** that would use a pencil and paper to solve some problem. Turing tried to decompose the operations of such a 'human computer' into a number of simple steps. He came to the conclusion that

- Any such step would consist of **erasing** a symbol on the paper at the tip of the pencil and **writing** a new one in its place.
- The decision as to which symbol should be written and which symbol should be at the tip of the pencil next would depend only
  - on the **symbol** currently at the tip of the pencil,
  - and the **'state of mind'** of the 'human computer'.

Based on these assumptions, Turing suggested a formal model of computation, known now as a **Turing machine**

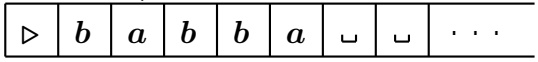
Turing machine

A **theoretical model** for computation, intended to capture the capabilities of **any procedure** or **program**



Finite control device:
at any moment can be in one of the **states**

Head: can **read** and **write**, move **left** and **right**



Input/output tape: it is divided into cells;
each cell may contain a symbol or be blank (␣).

How a Turing machine works

- The machine is supplied with input by inscribing the input string on the tape cells at the left end of the tape. The rest of the tape initially contains blank (␣) symbols. The head scans the leftmost cell of the input.

- At regular time intervals the machine performs two functions in a way
  dependent on the current state of its control device and
  the tape symbol currently scanned by the head:

  (1) Puts the control device in a new state.

  (2) Either (i) writes a symbol in the tape cell currently scanned,
  replacing the one already there

  or (ii) moves the head one tape cell to the left or right.

States and the tape of a Turing machine

States: represent the finite control device. There is always an **initial state** (s), where all computations of the machine start. Since the machine can write on the tape, it can leave an answer (output) on the tape at the end of computation. Therefore we don't need to provide special accepting states. On the other hand, since the head now can move back and forth, we do need a special **halting state** (h) to indicate the end of computation.

Tape: The tape has a left end and can be extended indefinitely to the right.

The leftmost cell contains a special marker $\triangleright$.

When the head scans this symbol, it **always** moves to the right.

(This way the machine never attempts to 'fall' from the left end of the tape.)

Transition function

It is the most important part of a Turing machine that describes its behaviour.

It can be given as a **transition** or **'instruction' table**:

Current state	Symbol scanned	Next state	Task
s	$\sqcup$	h	$\sqcup$
s	a	q	b
q	$\sqcup$	s	$\leftarrow$
s	$\triangleright$	s	$\rightarrow$
q	$\triangleright$	q	$\rightarrow$
q	a	h	a

In the 'Task' column a symbol like a (or $\sqcup$) means

'*replace the scanned symbol on the tape with* a (*or* $\sqcup$)'

$\rightarrow$ and $\leftarrow$ mean, respectively,

'*move the head one cell to the right*' or '*left*'

Turing machine: example 1

States of machine M_{eraser}: s, h, q.

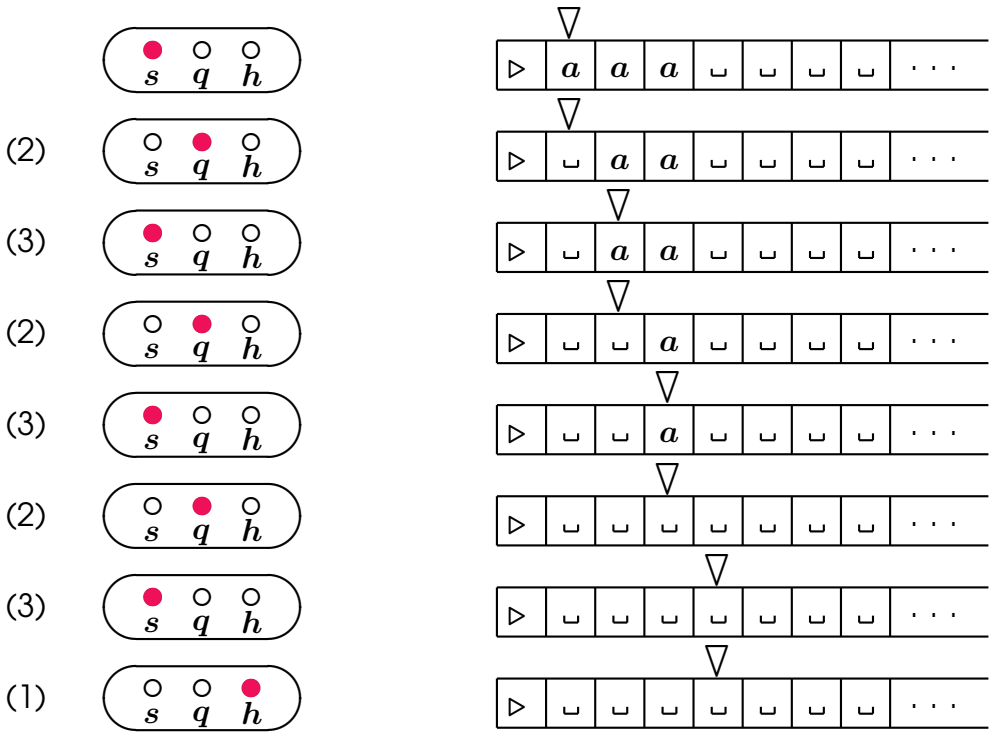
Symbols that can be written on the tape: $\triangleright, \sqcup, a$.

Transition table:

(1)	s	$\sqcup$	h	$\sqcup$
(2)	s	a	q	$\sqcup$
(3)	q	$\sqcup$	s	$\rightarrow$
(4)	s	$\triangleright$	s	$\rightarrow$
(5)	q	$\triangleright$	s	$\rightarrow$
(6)	q	a	h	a

- for each pair (non-halting state, symbol) there is a unique line in the table
- there are no lines starting with h
- if symbol $\triangleright$ is scanned then $\rightarrow$ should be the task to perform

How machine M_{eraser} works



Turing machine: formal definition

A **Turing machine** is a quintuple $M = (Q, \Sigma, \delta, s, h)$ where

- Q is a finite set of **states**, containing the **initial state** s and the **halting state** h,
- Σ is a finite set of symbols, the **tape alphabet**
  (containing the **left end marker** $\triangleright$ and the **blank** $\sqcup$, but **not** $\rightarrow$ and $\leftarrow$),
- δ is the **transition function** mapping
  each pair in $(Q - \{h\}) \times \Sigma$ to a pair in $Q \times ((\Sigma - \{\triangleright\}) \cup \{\rightarrow, \leftarrow\})$
  such that for all non-halting states q, $\delta(q, \triangleright) = (p, \rightarrow)$ for some state p.
  (This last bit guarantees that if M scans $\triangleright$ then the head always moves to the right.)

(non-halting state, tape alphabet symbol) $\overset{\delta}{\mapsto}$ (state, task)

Configurations of a Turing machine

As a Turing machine works, changes occur in the current state, the current tape contents, and the current head location.

A setting of these three items is called a **configuration** of the machine.

Since the input is always finite and the machine can move its head only one cell at a time, after a finite number of steps only finitely many tape cells can contain non-blank symbols. So we can represent a configuration by a pair like

$$(q,\ \triangleright ab\sqcup b\underline{a}a\sqcup\sqcup b)$$

where q is the current state, the current tape contents starts with $\triangleright ab\sqcup b\underline{a}a\sqcup\sqcup b$ and then all blank, and the head is currently scanning the underlined symbol.

A configuration of the form $(h, \dots)$ is called a **halting configuration**.

One step of a Turing machine

Say that configuration C_1 **yields** configuration C_2 **in one step** if the transition table shows that the Turing machine can 'legally go' from C_1 to C_2.

For example,

$(q_5,\ \triangleright u a \underline{b} v)$ yields $(q_7,\ \triangleright u \underline{a} b v)$ in one step

if the transition table of the machine contains the line

q_5	b	q_7	$\leftarrow$

$(s,\ \triangleright x y \underline{\sqcup} z z)$ yields $(q_3,\ \triangleright x y \underline{u} z z)$ in one step

if the transition table of the machine contains the line

s	$\sqcup$	q_3	u

Turing machine computations

The **computation of a Turing machine M on input word w** is the unique sequence of configurations

$$C_0, C_1, C_2, \ldots$$

such that

- C_0 is of the form $(s, \triangleright w)$ (where s is the initial state of M) and the head scans the leftmost symbol of w, and
- every C_i yields C_{i+1} in one step.

The Turing machine M **terminates** (or **halts**) **on input** w if there is a **finite** computation $C_0, C_1, \ldots, C_n$ on w such that

- C_n is a halting configuration.

We say that such a computation is of **length** n or has n **steps**

A computation of M_{eraser}

Recall (pages 6–7) that this machine just **'erases'** the input string from the tape.

For example, having started with the input aaa, M_{eraser} executes a cycle of using transitions (2) and (3) three times. When the tape is all blank, it applies transition (1) to halt.

The seven-step halting computation of M_{eraser} on input aaa is as follows:

$$(s, \triangleright\underline{a}aa), \ (q, \triangleright\underline{\sqcup}aa), \ (s, \triangleright\sqcup\underline{a}a), \ (q, \triangleright\sqcup\underline{\sqcup}a), \ (s, \triangleright\sqcup\sqcup\underline{a}), \ (q, \triangleright\sqcup\sqcup\underline{\sqcup}),$$

$$(s, \triangleright\sqcup\sqcup\sqcup\underline{\sqcup}), \ (h, \triangleright\sqcup\sqcup\sqcup\underline{\sqcup})$$

Turing machines are deterministic

Observe that for each non-halting configuration C of a Turing machine M, there is a **unique** configuration C' such that C yields C' in one step.
So starting from a non-halting configuration C_0,
the machine M has a **unique** computation

$$C_0, C_1, C_2, \ldots$$

This computation maybe **infinite** (when, starting with C_0,
M does not terminate).

But if it ends with some configuration C_n, then C_n **must** be a halting configuration. Otherwise, there is always a line in the transition table of M that says how to continue (so a Turing machine never gets stuck).

Turing machine: example 2

Consider the Turing machine $M_2 = (Q, \Sigma, \delta, s, h)$ where

$$Q = \{s, h, q\}, \quad \Sigma = \{\triangleright, \sqcup, \square, \diamond\}$$ and

δ is given by the following transition table:

	s	$\sqcup$	q	$\square$
	s	$\square$	q	$\square$
	s	$\diamond$	h	$\square$
	s	$\triangleright$	s	$\rightarrow$
$(\bullet)$	q	$\sqcup$	q	$\leftarrow$
$(*)$	q	$\square$	q	$\rightarrow$
	q	$\diamond$	q	$\rightarrow$
	q	$\triangleright$	s	$\rightarrow$

Example 2 (cont.)

How the machine on the previous slide works on input $\square\diamond\square$:

- First, it changes its state to q.
- Then the head goes to the right end of the input word. When it reaches the first blank, the head starts to move back and forth indefinitely, alternating between transitions ($\bullet$) and ($*$).
- Thus the machine **never terminates** on input $\square\diamond\square$.

$$(s, \triangleright\underline{\square}\diamond\square),\ (q, \triangleright\underline{\square}\diamond\square),\ (q, \triangleright\square\underline{\diamond}\square),\ (q, \triangleright\square\diamond\underline{\square}),\ (q, \triangleright\square\diamond\square\underline{\sqcup}),$$
$$(q, \triangleright\square\diamond\underline{\square}),\ (q, \triangleright\square\diamond\square\underline{\sqcup}),\ (q, \triangleright\square\diamond\underline{\square}),\ (q, \triangleright\square\diamond\square\underline{\sqcup}),\ \ldots$$

There can be Turing machine computations that never stop

Example 2 (cont.)

How the machine on the previous slide works on input ◇□□□:

It stops in one step

$$(s, \triangleright\underline{\diamond}\square\square\square),\ (h, \triangleright\underline{\square}\square\square\square)$$

It can happen that

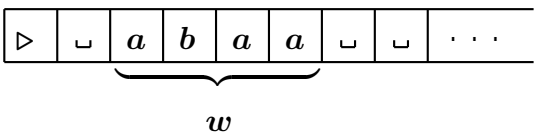
a Turing machine terminates on some inputs,

while runs forever on other inputs

Left-shifting machine

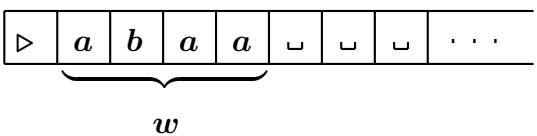
Problem: design a Turing machine that does the following:

For any string w of a's and b's, the machine should start with the tape like



and then should **'shift'** the string w one position to the left.

That is, it should end up with the tape like



Left-shifting machine: implementation

The above 'left-shift' can be implemented by a loop that, on every iteration,

(1) moves the head one cell to the right;

(2) if the symbol observed by the head is not blank then memorises it with the help of a new state (say, q_a or q_b) and temporarily 'replaces' it by a blank symbol;

(3) moves the head one cell back to the left (during which it shouldn't forget the memorised symbol);

(4) depending on the symbol that has been memorised, it writes a or b, and

(5) moves the head one cell to the right.

If at any round the symbol observed in step (2) is blank, then the machine halts.

Left-shifting machine: transition table

(1)	s	$\sqcup$	p	$\rightarrow$
(2)	p	a	q_a	$\sqcup$
	p	b	q_b	$\sqcup$
(3)	q_a	$\sqcup$	r_a	$\leftarrow$
	q_b	$\sqcup$	r_b	$\leftarrow$
(4)	r_a	$\sqcup$	t	a
	r_b	$\sqcup$	t	b
(5)	t	a	s	$\rightarrow$
	t	b	s	$\rightarrow$
	p	$\sqcup$	h	$\sqcup$

. . . the rest is arbitrary (but $\boxed{\ldots}\boxed{\triangleright}$ should always go to $\boxed{\ldots}\boxed{\rightarrow}$)

Left-shifting machine: test computation

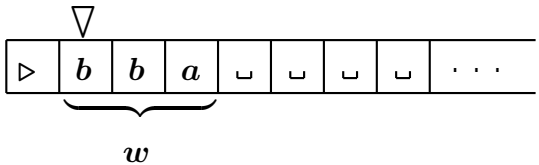
The 22-step computation of the left-shifting machine on the input word $\text{␣}\,abaa$:

$$(s, \triangleright\underline{\text{␣}}abaa),\ (p, \triangleright\text{␣}\underline{a}baa),\ (q_a, \triangleright\text{␣␣}baa),\ (r_a, \triangleright\underline{\text{␣}}\text{␣}baa),\ (t, \triangleright\underline{a}\text{␣}baa),$$
$$(s, \triangleright a\underline{\text{␣}}baa),\ (p, \triangleright a\text{␣}\underline{b}aa),\ (q_b, \triangleright a\text{␣␣}aa),\ (r_b, \triangleright a\underline{\text{␣}}\text{␣}aa),\ (t, \triangleright a\underline{b}\text{␣}aa),$$
$$(s, \triangleright ab\underline{\text{␣}}aa),\ (p, \triangleright ab\text{␣}\underline{a}a),\ (q_a, \triangleright ab\text{␣␣}a),\ (r_a, \triangleright ab\underline{\text{␣}}\text{␣}a),\ (t, \triangleright ab\underline{a}\text{␣}a),$$
$$(s, \triangleright aba\underline{\text{␣}}a),\ (p, \triangleright aba\text{␣}\underline{a}),\ (q_a, \triangleright aba\text{␣␣}),\ (r_a, \triangleright aba\underline{\text{␣}}\text{␣}),\ (t, \triangleright aba\underline{a}\text{␣}),$$
$$(s, \triangleright abaa\underline{\text{␣}}),\ (p, \triangleright abaa\text{␣␣}),\ (h, \triangleright abaa\text{␣␣})$$

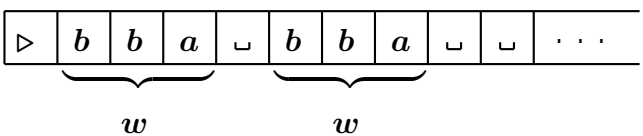
Copying machine

Problem: design a Turing machine that does the following.

For *any* word w of a's and b's, the machine should start with w being written on the left end of the tape and the head scanning the leftmost symbol of w,



and then it should transform the tape contents to $w␣w$. That is, it should end up (for the example above) with the tape



Copying machine: implementation idea

This can be implemented by a loop that, on every iteration,

(1) if the symbol observed by the head is not blank then memorises it with the help of a new state (say, q_a or q_b) and temporarily 'replaces' it by a blank symbol;

(2) moves the head to the right until it reaches the second blank cell (during which it shouldn't forget the memorised symbol);

(3) depending on the symbol that has been memorised, it writes a or b,

(4) moves the head to the left until it reaches the second blank cell (it should keep remembering the memorised symbol);

(5) restores the memorised symbol;

(6) moves the head one step to the right.

If at any round the symbol observed in step (1) is blank, then the machine halts.

Copying machine: transition table

(1)	s	a	q_a	$\sqcup$
	s	b	q_b	$\sqcup$
(2)	q_a	$\sqcup$	r_a	$\rightarrow$
	q_b	$\sqcup$	r_b	$\rightarrow$
	r_a	a	r_a	$\rightarrow$
	r_a	b	r_a	$\rightarrow$
	r_a	$\sqcup$	p_a	$\rightarrow$
	p_a	a	p_a	$\rightarrow$
	p_a	b	p_a	$\rightarrow$

(2 cont.)	r_b	a	r_b	$\rightarrow$
	r_b	b	r_b	$\rightarrow$
	r_b	$\sqcup$	p_b	$\rightarrow$
	p_b	a	p_b	$\rightarrow$
	p_b	b	p_b	$\rightarrow$
(3)	p_a	$\sqcup$	t_a	a
	p_b	$\sqcup$	t_b	b

Copying machine: transition table (cont.)

(4)	t_a	a	t_a	$\leftarrow$
	t_a	b	t_a	$\leftarrow$
	t_a	$\sqcup$	n_a	$\leftarrow$
	t_b	a	t_b	$\leftarrow$
	t_b	b	t_b	$\leftarrow$
	t_b	$\sqcup$	n_b	$\leftarrow$
	n_a	a	n_a	$\leftarrow$
	n_a	b	n_a	$\leftarrow$
	n_b	a	n_b	$\leftarrow$
	n_b	b	n_b	$\leftarrow$

(5)	n_a	$\sqcup$	m	a
	n_b	$\sqcup$	m	b
(6)	m	a	s	$\rightarrow$
	m	b	s	$\rightarrow$
	s	$\sqcup$	h	$\sqcup$

... the rest is arbitrary (but $\boxed{\ldots}\boxed{\triangleright}$ should always go to $\boxed{\ldots}\boxed{\rightarrow}$).

Copying machine: test computation

The 37-step computation of the copying machine on input word bba:

$$
\begin{bmatrix}
(s, \triangleright\underline{b}bba),\ (q_b, \triangleright\underline{\sqcup}ba),\ (r_b, \triangleright\sqcup\underline{b}a),\ (r_b, \triangleright\sqcup b\underline{a}),\ (r_b, \triangleright\sqcup ba\underline{\sqcup}),\\
(p_b, \triangleright\sqcup ba\sqcup\underline{\sqcup}),\ (t_b, \triangleright\sqcup ba\sqcup\underline{b}),\ (t_b, \triangleright\sqcup ba\underline{\sqcup}b),\ (n_b, \triangleright\sqcup b\underline{a}\sqcup b),\\
(n_b, \triangleright\sqcup\underline{b}a\sqcup b),\ (n_b, \triangleright\underline{\sqcup}ba\sqcup b),\ (m, \triangleright\underline{b}ba\sqcup b),
\end{bmatrix}
$$

$$
\begin{bmatrix}
(s, \triangleright b\underline{b}a\sqcup b),\ (q_b, \triangleright b\underline{\sqcup}a\sqcup b),\ (r_b, \triangleright b\sqcup\underline{a}\sqcup b),\ (r_b, \triangleright b\sqcup a\underline{\sqcup}b),\\
(p_b, \triangleright b\sqcup a\sqcup\underline{b}),\ (p_b, \triangleright b\sqcup a\sqcup b\underline{\sqcup}),\ (t_b, \triangleright b\sqcup a\sqcup b\underline{b}),\ (t_b, \triangleright b\sqcup a\sqcup\underline{b}b),\\
(t_b, \triangleright b\sqcup a\underline{\sqcup}bb),\ (n_b, \triangleright b\sqcup\underline{a}\sqcup bb),\ (n_b, \triangleright b\underline{\sqcup}a\sqcup bb),\ (m, \triangleright b\underline{b}a\sqcup bb),
\end{bmatrix}
$$

$$
\begin{bmatrix}
(s, \triangleright bb\underline{a}\sqcup bb),\ (q_a, \triangleright bb\underline{\sqcup}\sqcup bb),\ (r_a, \triangleright bb\sqcup\underline{\sqcup}bb),\ (p_a, \triangleright bb\sqcup\sqcup\underline{b}b),\\
(p_a, \triangleright bb\sqcup\sqcup b\underline{b}),\ (p_a, \triangleright bb\sqcup\sqcup bb\underline{\sqcup}),\ (t_a, \triangleright bb\sqcup\sqcup bb\underline{a}),\ (t_a, \triangleright bb\sqcup\sqcup b\underline{b}a),\\
(t_a, \triangleright bb\sqcup\sqcup\underline{b}ba),\ (t_a, \triangleright bb\sqcup\underline{\sqcup}bba),\ (n_a, \triangleright bb\underline{\sqcup}\sqcup bba),\ (m, \triangleright bb\underline{a}\sqcup bba),
\end{bmatrix}
$$

$$
(s, \triangleright bba\underline{\sqcup}bba),\ (h, \triangleright bba\underline{\sqcup}bba)
$$

Finite automata and Turing machines

Main differences between finite automata and Turing machines:

(1) A Turing machine can both write on the tape and read from it.

(2) The read/write head can move both to the left and to the right.

(3) After reaching an accepting state, a finite automaton can continue its computation. Reaching the halting state of a Turing machine takes immediate effect.

(4) DFA and NFA always stop working after a finite number of steps (though NFAs can get stuck). PDA and Turing machines can loop.

Finite automata can be regarded as special cases of Turing machines. In other words, every finite automaton can be 'simulated' by a Turing machine (but **not** the other way round).

Computational tasks

Turing machines are not only simple data processing devices.

They can perform many important computational tasks such as

- compute functions on strings
- recognise languages
- compute arithmetic functions
- ...

A function $f : \Sigma^* \to \Sigma^*$, for an alphabet Σ, is called a **function on Σ-strings**

(in other words, f maps each string w over Σ to a string $f(w)$ over Σ)

(1) $\Sigma = \{a, b\}$; for each word w over Σ, let $f(w) = wa$

Then $f(aaabb) = aaabba$, $f(\varepsilon) = a$, $f(ba) = baa$, ...

(2) $\Sigma = \{\square, \diamond\}$; for each word w over Σ, let $f(w) = \begin{cases} w\square w, & \text{if } |w| \text{ is even} \\ w, & \text{if } |w| \text{ is odd} \end{cases}$

Then $f(\square\diamond) = \square\diamond\square\square\diamond$, $f(\varepsilon) = \square$, $f(\diamond) = \diamond$, ...

Computing functions on strings

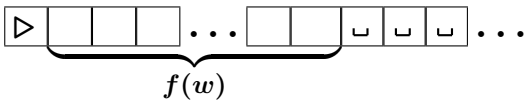
Let Σ be an alphabet and f a function on Σ-strings

(that is, f maps each string w over Σ to a string $f(w)$ over Σ).

Let M be a Turing machine with tape alphabet $\Sigma \cup \{\triangleright, \sqcup\}$.

Say that Turing machine M **computes the function** f if, for every string w over Σ

- M halts on input w
- after halting the tape looks like



that is, $f(w)$ is the **output** of M on input w

A function f on strings is called **Turing computable** if there exists a Turing machine M that computes it

Turing computable functions: example 1

Let $\Sigma = \{a, b\}$.

Consider the function $f(w) = wa$ defined on all strings over Σ.

The Turing machine M computing f moves its head to the right until the first blank, replaces the blank by a and halts.

s	a	s	$\rightarrow$
s	b	s	$\rightarrow$
s	$\triangleright$	s	$\rightarrow$
s	$\sqcup$	h	a

Observe that M halts on every input string w and delivers the output wa.

Thus M computes f

Turing machine computing $f(w) = wa$: test

The 5-step computation of the machine on input word $w = baab$:

$$(s, \triangleright\underline{b}aab),\ (s, \triangleright b\underline{a}ab),\ (s, \triangleright ba\underline{a}b),\ (s, \triangleright baab\underline{\sqcup}),\ (h, \triangleright \underbrace{baaba}_{f(w)=wa}\underline{a})$$

The 6-step computation of the machine on input word $w = aabbb$:

$$(s, \triangleright\underline{a}abbb),\ (s, \triangleright a\underline{a}bbb),\ (s, \triangleright aa\underline{b}bb),\ (s, \triangleright aab\underline{b}b),\ (s, \triangleright aabb\underline{b}),\ (s, \triangleright aabbb\underline{\sqcup}),$$

$$(h, \triangleright \underbrace{aabbb}_{f(w)=wa}\underline{a})$$

Turing computable functions: example 2

Let $\Sigma = \{a, b\}$.

Consider the function $f(w) = ww$ defined on all strings over Σ.

The Turing machine M computing f must transform any string of a's and b's into the 'same string written twice,' and then halt

Recall the '*copying machine*' on pages 21–23.

It transforms w to $w \sqcup w$, and when it halts, its head scans the blank cell in the middle.

This machine can be augmented to output ww instead of $w \sqcup w$.

To achieve this, after the copying machine terminates, start to operate the '*left-shifting machine*' (see pages 17–20).

Turing machine computing $f(w) = ww$: test

The (37+22)-step computation of the machine on input $w = abba$:

$(s_{copy}, \triangleright\underline{a}bba)$, copying machine, $(h_{copy}, \triangleright abba\underline{\sqcup}abba)$,

... we continue with the left-shifting machine, by identifying the halting-state h_{copy} with the initial state $s_{leftshift}$

... $(h_{leftshift}, \triangleright \underbrace{abbaabba}_{f(w)=ww\text{: the output}} \underline{\sqcup}\sqcup)$