I. Turing Machines & Classical Computability.

Alan Turing
(1912-1954)

1. Turing Machines

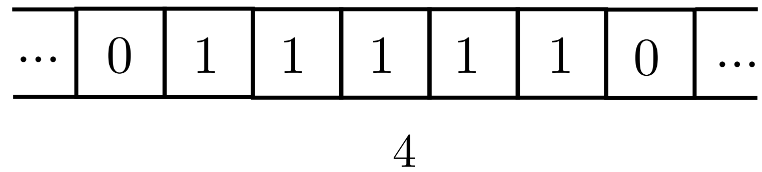
- A Turing machine (TM) consists of (Turing 1936):

1. *An unbounded tape*. Divided into squares, each square containing a symbol from a finite alphabet $\{q_0, q_1, ..., q_n\}$.
2. *A read/write scanner*. Programmed with a finite list of states $\{s_0, ..., s_m\}$.
3. *A program*. Consists of a finite sequence of transition rules. Each rule consists of a 4-tuple ⟨*initial state*, *initial symbol*, *final state*, *action*⟩. For initial state and initial symbol s_i, q_j there are 3 possible actions, afterwhich the final state s_ℓ is entered:
   (a) Replace initial symbol with q_k. $\langle s_i, q_j, s_\ell, q_k\rangle$.
   (b) Move one square left. $\langle s_i, q_j, s_\ell, \ll\rangle$.
   (b) Move one square right. $\langle s_i, q_j, s_\ell, \gg\rangle$.

- A TM halts when no unique transition rule is available to it.

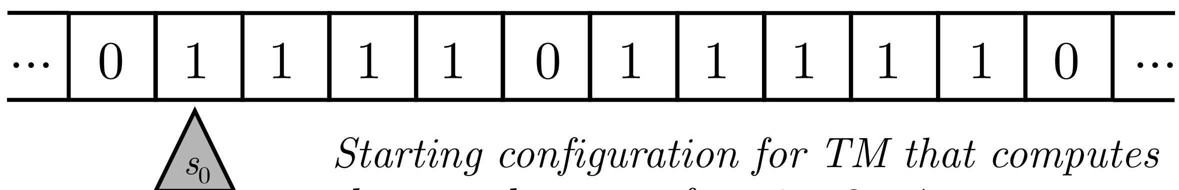
Conventions

(i) Represent the number n by a block of $n+1$ "1"s.



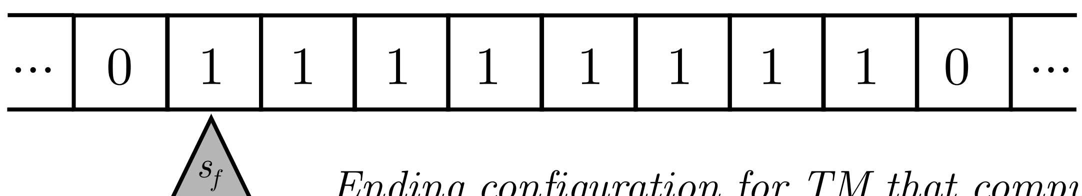
(ii) *Starting configuration*:

- TM starts in lowest-numbered state.
- Scanner starts at leftmost "1" of input block, with "0" to left.
- For computing functions with n arguments, input block consists of n blocks of "1"s separated by a "0", each block encoding an argument.



Starting configuration for TM that computes the two-place sum function $3+4$.

(iii) *Ending configuration*: Scanner ends at leftmost "1" of output block, with "0" to left.



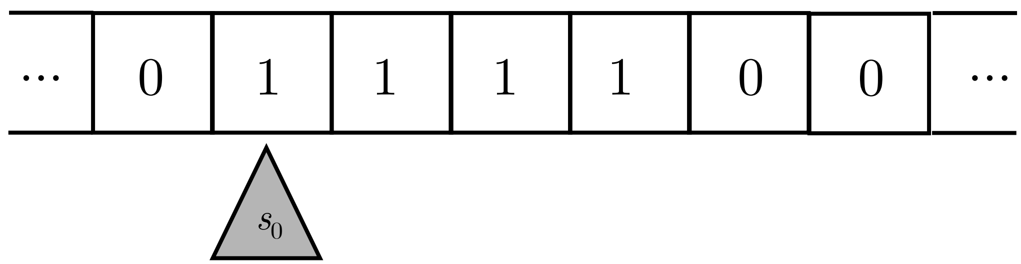
Ending configuration for TM that computes the two-place sum function $3+4$.

Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0, 1, s_0, \gg\rangle, \langle s_0, 0, s_1, 1\rangle, \langle s_1, 1, s_1, \ll\rangle, \langle s_1, 0, s_2, \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Start.

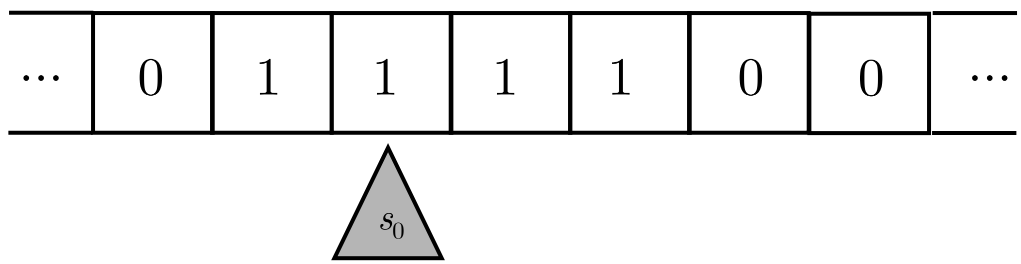


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0,\ 1,\ s_0,\ \gg\rangle,\ \langle s_0,\ 0,\ s_1,\ 1\rangle,\ \langle s_1,\ 1,\ s_1,\ \ll\rangle,\ \langle s_1,\ 0,\ s_2,\ \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 1.

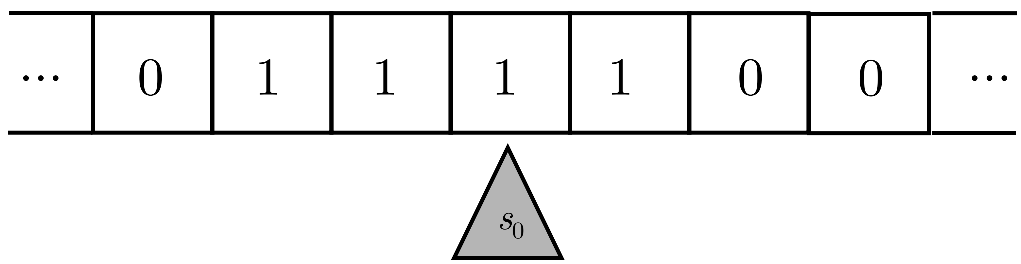


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0,\ 1,\ s_0,\ \gg\rangle,\ \langle s_0,\ 0,\ s_1,\ 1\rangle,\ \langle s_1,\ 1,\ s_1,\ \ll\rangle,\ \langle s_1,\ 0,\ s_2,\ \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 2.

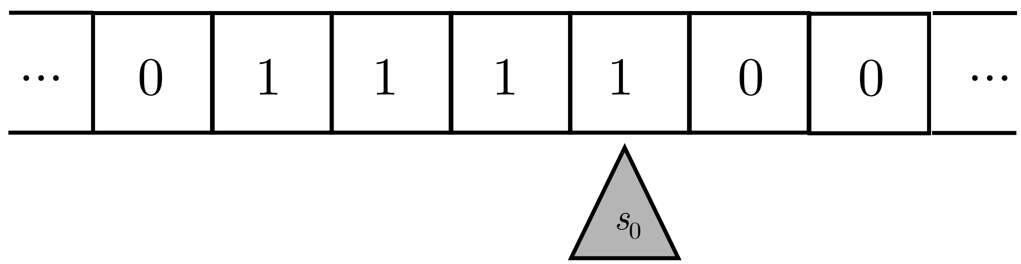


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0,\ 1,\ s_0,\ \gg\rangle,\ \langle s_0,\ 0,\ s_1,\ 1\rangle,\ \langle s_1,\ 1,\ s_1,\ \ll\rangle,\ \langle s_1,\ 0,\ s_2,\ \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 3.

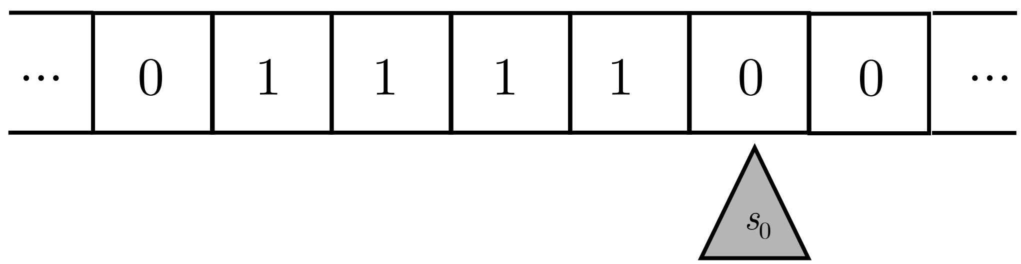


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0,\ 1,\ s_0,\ \gg\rangle,\ \langle s_0,\ 0,\ s_1,\ 1\rangle,\ \langle s_1,\ 1,\ s_1,\ \ll\rangle,\ \langle s_1,\ 0,\ s_2,\ \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 4.

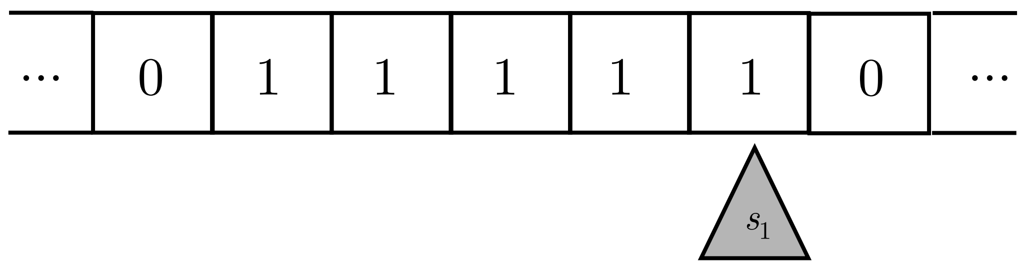


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0,\ 1,\ s_0,\ \gg\rangle,\ \langle s_0,\ 0,\ s_1,\ 1\rangle,\ \langle s_1,\ 1,\ s_1,\ \ll\rangle,\ \langle s_1,\ 0,\ s_2,\ \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 5.

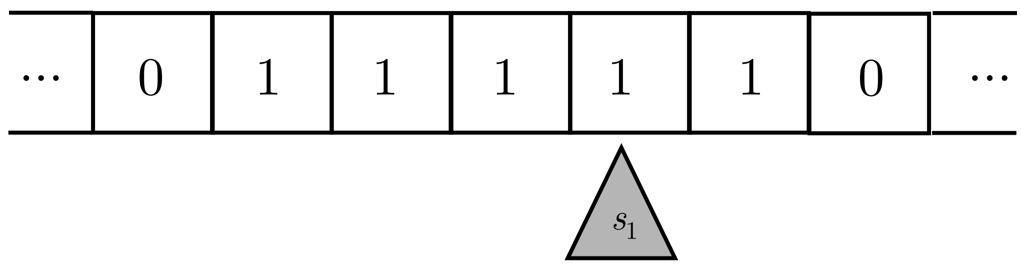


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0, 1, s_0, \gg\rangle, \langle s_0, 0, s_1, 1\rangle, \langle s_1, 1, s_1, \ll\rangle, \langle s_1, 0, s_2, \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 6.

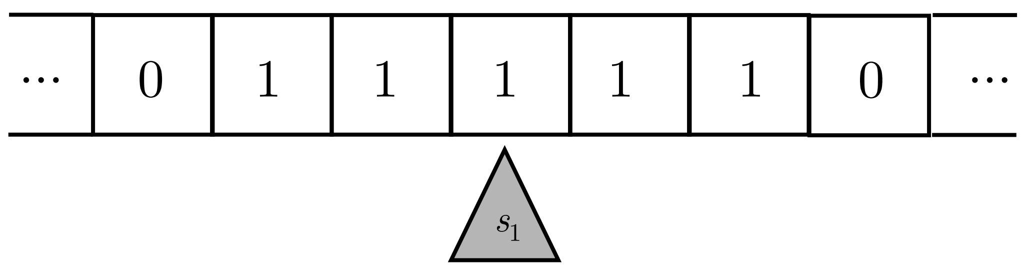


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0,\, 1,\, s_0,\, \gg\rangle,\ \langle s_0,\, 0,\, s_1,\, 1\rangle,\ \langle s_1,\, 1,\, s_1,\, \ll\rangle,\ \langle s_1,\, 0,\, s_2,\, \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 7.

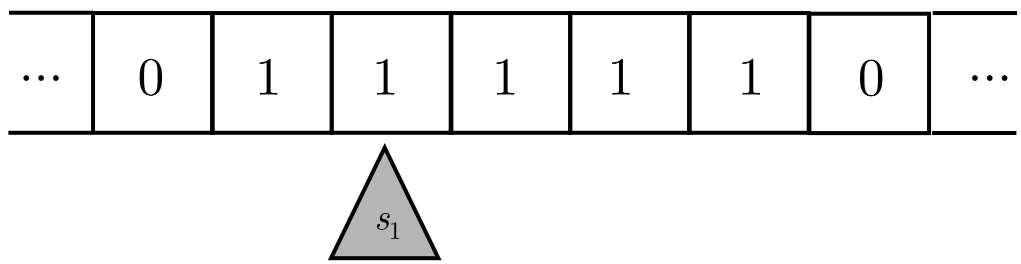


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0,\ 1,\ s_0,\ \gg\rangle,\ \langle s_0,\ 0,\ s_1,\ 1\rangle,\ \langle s_1,\ 1,\ s_1,\ \ll\rangle,\ \langle s_1,\ 0,\ s_2,\ \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 8.

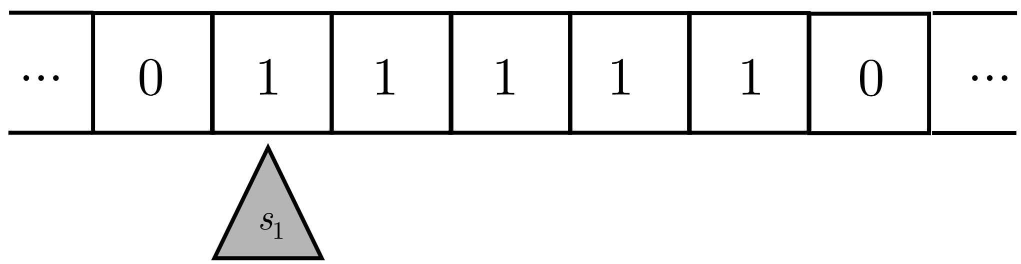


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0,\ 1,\ s_0,\ \gg\rangle,\ \langle s_0,\ 0,\ s_1,\ 1\rangle,\ \langle s_1,\ 1,\ s_1,\ \ll\rangle,\ \langle s_1,\ 0,\ s_2,\ \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 9.

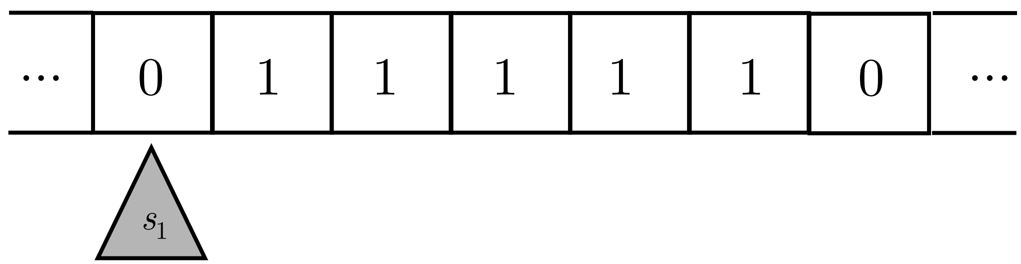


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0,\ 1,\ s_0,\ \gg\rangle,\ \langle s_0,\ 0,\ s_1,\ 1\rangle,\ \langle s_1,\ 1,\ s_1,\ \ll\rangle,\ \langle s_1,\ 0,\ s_2,\ \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

Step 10.

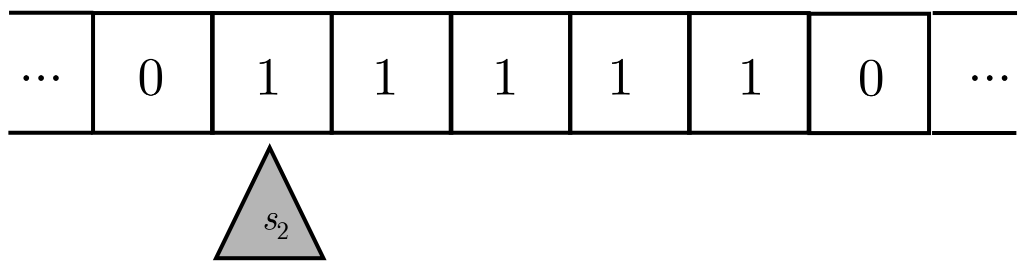


Example 1: TM that computes successor function $f(n) = n + 1$.

$$\langle s_0, 1, s_0, \gg\rangle, \langle s_0, 0, s_1, 1\rangle, \langle s_1, 1, s_1, \ll\rangle, \langle s_1, 0, s_2, \gg\rangle$$

- Stays in state s_0 and scans right until initial block of "1"s (input) is scanned.
- Replaces "0" at end of input block with "1".
- Enters state s_1 and scans back left to beginning of block.
- When "0" is reached at beginning, enters state s_2 and scans right.
- Halts in standard ending configuration (no rule can be followed in state s_2).

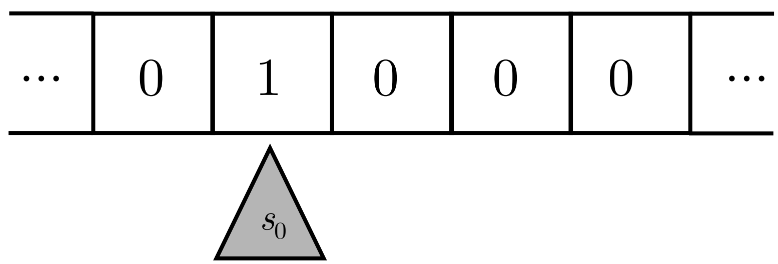
End.



Example 2: TM copier.

$\langle s_0, 1, s_0, A\rangle$	$\langle s_1, 1, s_1, \gg\rangle$	$\langle s_3, 1, s_3, \ll\rangle$	$\langle s_5, A, s_5, 1\rangle$
$\langle s_0, A, s_1, \gg\rangle$	$\langle s_1, 0, s_2, \gg\rangle$	$\langle s_3, 0, s_4, \ll\rangle$	$\langle s_5, 1, s_5, \ll\rangle$
$\langle s_0, 0, s_5, \ll\rangle$	$\langle s_2, 1, s_2, \gg\rangle$	$\langle s_4, 1, s_4, \ll\rangle$	$\langle s_5, 0, s_6, \gg\rangle$
	$\langle s_2, 0, s_3, 1\rangle$	$\langle s_4, A, s_0, \gg\rangle$	

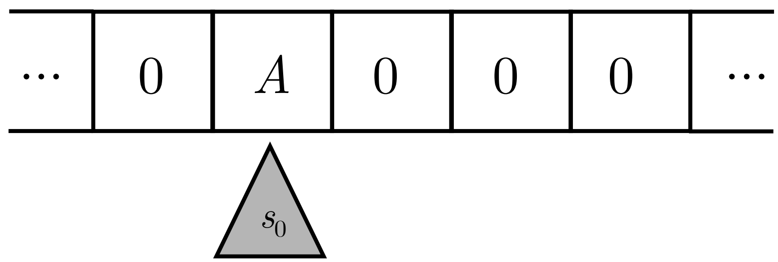
Start.



Example 2: TM copier.

$$\begin{array}{llll}
\langle s_0,\ 1,\ s_0,\ A\rangle & \langle s_1,\ 1,\ s_1,\ \gg\rangle & \langle s_3,\ 1,\ s_3,\ \ll\rangle & \langle s_5,\ A,\ s_5,\ 1\rangle \\
\langle s_0,\ A,\ s_1,\ \gg\rangle & \langle s_1,\ 0,\ s_2,\ \gg\rangle & \langle s_3,\ 0,\ s_4,\ \ll\rangle & \langle s_5,\ 1,\ s_5,\ \ll\rangle \\
\langle s_0,\ 0,\ s_5,\ \ll\rangle & \langle s_2,\ 1,\ s_2,\ \gg\rangle & \langle s_4,\ 1,\ s_4,\ \ll\rangle & \langle s_5,\ 0,\ s_6,\ \gg\rangle \\
 & \langle s_2,\ 0,\ s_3,\ 1\rangle & \langle s_4,\ A,\ s_0,\ \gg\rangle &
\end{array}$$

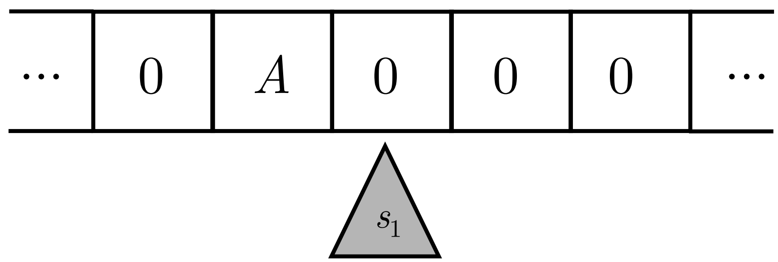
Step 1.



Example 2: TM copier.

$\langle s_0, 1, s_0, A\rangle$	$\langle s_1, 1, s_1, \gg\rangle$	$\langle s_3, 1, s_3, \ll\rangle$	$\langle s_5, A, s_5, 1\rangle$
$\langle s_0, A, s_1, \gg\rangle$	$\langle s_1, 0, s_2, \gg\rangle$	$\langle s_3, 0, s_4, \ll\rangle$	$\langle s_5, 1, s_5, \ll\rangle$
$\langle s_0, 0, s_5, \ll\rangle$	$\langle s_2, 1, s_2, \gg\rangle$	$\langle s_4, 1, s_4, \ll\rangle$	$\langle s_5, 0, s_6, \gg\rangle$
	$\langle s_2, 0, s_3, 1\rangle$	$\langle s_4, A, s_0, \gg\rangle$	

Step 2.



<u>Example 2</u>: TM copier.

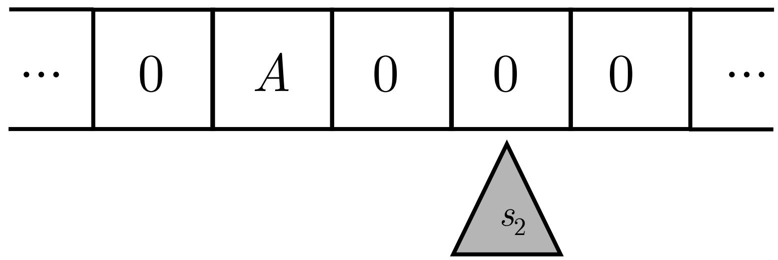
$$\langle s_0,\ 1,\ s_0,\ A\rangle$$
$$\langle s_0,\ A,\ s_1,\ \gg\rangle$$
$$\langle s_0,\ 0,\ s_5,\ \ll\rangle$$

$$\langle s_1,\ 1,\ s_1,\ \gg\rangle$$
$$\langle s_1,\ 0,\ s_2,\ \gg\rangle$$
$$\langle s_2,\ 1,\ s_2,\ \gg\rangle$$
$$\langle s_2,\ 0,\ s_3,\ 1\rangle$$

$$\langle s_3,\ 1,\ s_3,\ \ll\rangle$$
$$\langle s_3,\ 0,\ s_4,\ \ll\rangle$$
$$\langle s_4,\ 1,\ s_4,\ \ll\rangle$$
$$\langle s_4,\ A,\ s_0,\ \gg\rangle$$

$$\langle s_5,\ A,\ s_5,\ 1\rangle$$
$$\langle s_5,\ 1,\ s_5,\ \ll\rangle$$
$$\langle s_5,\ 0,\ s_6,\ \gg\rangle$$

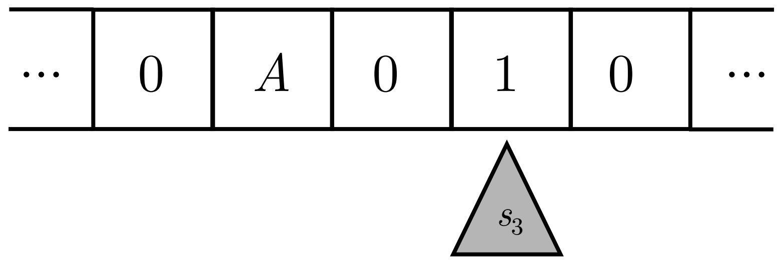
Step 3.



Example 2: TM copier.

$\langle s_0, 1, s_0, A\rangle$	$\langle s_1, 1, s_1, \gg\rangle$	$\langle s_3, 1, s_3, \ll\rangle$	$\langle s_5, A, s_5, 1\rangle$
$\langle s_0, A, s_1, \gg\rangle$	$\langle s_1, 0, s_2, \gg\rangle$	$\langle s_3, 0, s_4, \ll\rangle$	$\langle s_5, 1, s_5, \ll\rangle$
$\langle s_0, 0, s_5, \ll\rangle$	$\langle s_2, 1, s_2, \gg\rangle$	$\langle s_4, 1, s_4, \ll\rangle$	$\langle s_5, 0, s_6, \gg\rangle$
	$\langle s_2, 0, s_3, 1\rangle$	$\langle s_4, A, s_0, \gg\rangle$	

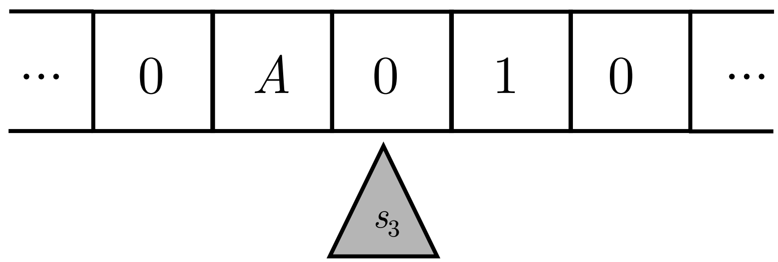
Step 4.



Example 2: TM copier.

$\langle s_0, 1, s_0, A\rangle$	$\langle s_1, 1, s_1, \gg\rangle$	$\langle s_3, 1, s_3, \ll\rangle$	$\langle s_5, A, s_5, 1\rangle$
$\langle s_0, A, s_1, \gg\rangle$	$\langle s_1, 0, s_2, \gg\rangle$	$\langle s_3, 0, s_4, \ll\rangle$	$\langle s_5, 1, s_5, \ll\rangle$
$\langle s_0, 0, s_5, \ll\rangle$	$\langle s_2, 1, s_2, \gg\rangle$	$\langle s_4, 1, s_4, \ll\rangle$	$\langle s_5, 0, s_6, \gg\rangle$
	$\langle s_2, 0, s_3, 1\rangle$	$\langle s_4, A, s_0, \gg\rangle$	

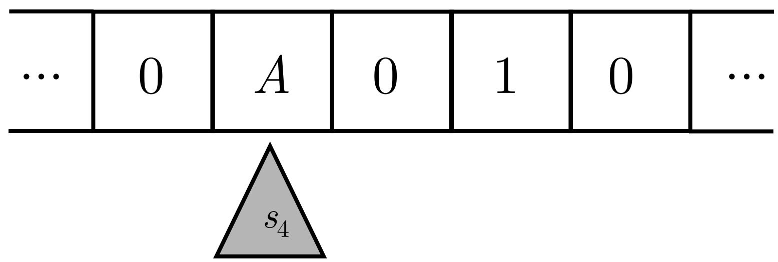
Step 5.



Example 2: TM copier.

$\langle s_0, 1, s_0, A \rangle$	$\langle s_1, 1, s_1, \gg \rangle$	$\langle s_3, 1, s_3, \ll \rangle$	$\langle s_5, A, s_5, 1 \rangle$
$\langle s_0, A, s_1, \gg \rangle$	$\langle s_1, 0, s_2, \gg \rangle$	$\langle s_3, 0, s_4, \ll \rangle$	$\langle s_5, 1, s_5, \ll \rangle$
$\langle s_0, 0, s_5, \ll \rangle$	$\langle s_2, 1, s_2, \gg \rangle$	$\langle s_4, 1, s_4, \ll \rangle$	$\langle s_5, 0, s_6, \gg \rangle$
	$\langle s_2, 0, s_3, 1 \rangle$	$\langle s_4, A, s_0, \gg \rangle$	

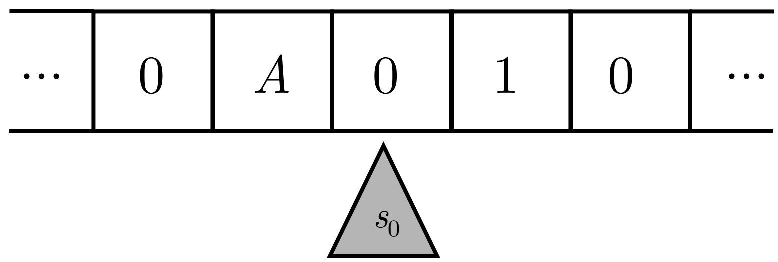
Step 6.



Example 2: TM copier.

$\langle s_0, 1, s_0, A\rangle$	$\langle s_1, 1, s_1, \gg\rangle$	$\langle s_3, 1, s_3, \ll\rangle$	$\langle s_5, A, s_5, 1\rangle$
$\langle s_0, A, s_1, \gg\rangle$	$\langle s_1, 0, s_2, \gg\rangle$	$\langle s_3, 0, s_4, \ll\rangle$	$\langle s_5, 1, s_5, \ll\rangle$
$\langle s_0, 0, s_5, \ll\rangle$	$\langle s_2, 1, s_2, \gg\rangle$	$\langle s_4, 1, s_4, \ll\rangle$	$\langle s_5, 0, s_6, \gg\rangle$
	$\langle s_2, 0, s_3, 1\rangle$	$\langle s_4, A, s_0, \gg\rangle$	

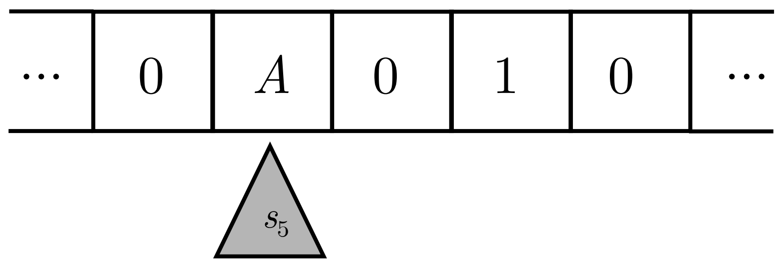
Step 7.



Example 2: TM copier.

$\langle s_0, 1, s_0, A\rangle$	$\langle s_1, 1, s_1, \gg\rangle$	$\langle s_3, 1, s_3, \ll\rangle$	$\langle s_5, A, s_5, 1\rangle$
$\langle s_0, A, s_1, \gg\rangle$	$\langle s_1, 0, s_2, \gg\rangle$	$\langle s_3, 0, s_4, \ll\rangle$	$\langle s_5, 1, s_5, \ll\rangle$
$\langle s_0, 0, s_5, \ll\rangle$	$\langle s_2, 1, s_2, \gg\rangle$	$\langle s_4, 1, s_4, \ll\rangle$	$\langle s_5, 0, s_6, \gg\rangle$
	$\langle s_2, 0, s_3, 1\rangle$	$\langle s_4, A, s_0, \gg\rangle$	

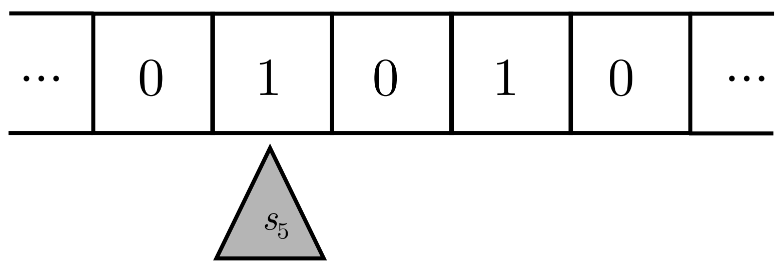
Step 8.



Example 2: TM copier.

$\langle s_0, 1, s_0, A \rangle$	$\langle s_1, 1, s_1, \gg \rangle$	$\langle s_3, 1, s_3, \ll \rangle$	$\langle s_5, A, s_5, 1 \rangle$
$\langle s_0, A, s_1, \gg \rangle$	$\langle s_1, 0, s_2, \gg \rangle$	$\langle s_3, 0, s_4, \ll \rangle$	$\langle s_5, 1, s_5, \ll \rangle$
$\langle s_0, 0, s_5, \ll \rangle$	$\langle s_2, 1, s_2, \gg \rangle$	$\langle s_4, 1, s_4, \ll \rangle$	$\langle s_5, 0, s_6, \gg \rangle$
	$\langle s_2, 0, s_3, 1 \rangle$	$\langle s_4, A, s_0, \gg \rangle$	

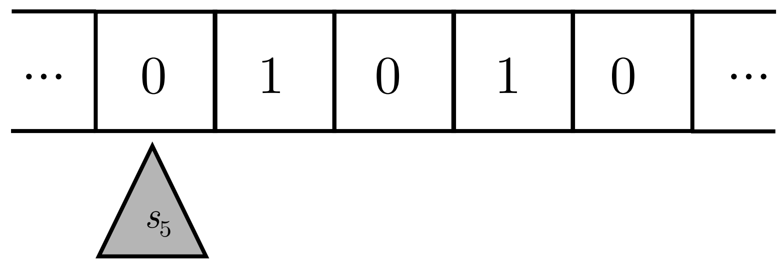
Step 9.



Example 2: TM copier.

$\langle s_0, 1, s_0, A\rangle$	$\langle s_1, 1, s_1, \gg\rangle$	$\langle s_3, 1, s_3, \ll\rangle$	$\langle s_5, A, s_5, 1\rangle$
$\langle s_0, A, s_1, \gg\rangle$	$\langle s_1, 0, s_2, \gg\rangle$	$\langle s_3, 0, s_4, \ll\rangle$	$\langle s_5, 1, s_5, \ll\rangle$
$\langle s_0, 0, s_5, \ll\rangle$	$\langle s_2, 1, s_2, \gg\rangle$	$\langle s_4, 1, s_4, \ll\rangle$	$\langle s_5, 0, s_6, \gg\rangle$
	$\langle s_2, 0, s_3, 1\rangle$	$\langle s_4, A, s_0, \gg\rangle$	

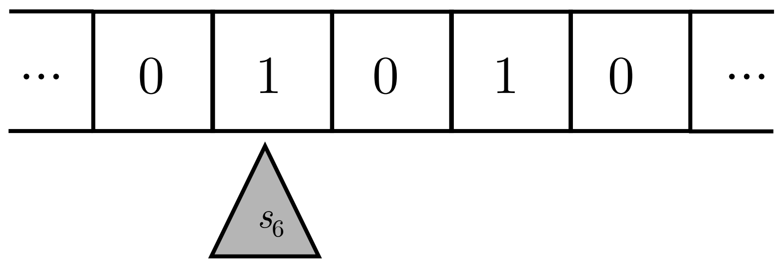
Step 10.



Example 2: TM copier.

$\langle s_0,\ 1,\ s_0,\ A\rangle$	$\langle s_1,\ 1,\ s_1,\ \gg\rangle$	$\langle s_3,\ 1,\ s_3,\ \ll\rangle$	$\langle s_5,\ A,\ s_5,\ 1\rangle$
$\langle s_0,\ A,\ s_1,\ \gg\rangle$	$\langle s_1,\ 0,\ s_2,\ \gg\rangle$	$\langle s_3,\ 0,\ s_4,\ \ll\rangle$	$\langle s_5,\ 1,\ s_5,\ \ll\rangle$
$\langle s_0,\ 0,\ s_5,\ \ll\rangle$	$\langle s_2,\ 1,\ s_2,\ \gg\rangle$	$\langle s_4,\ 1,\ s_4,\ \ll\rangle$	$\langle s_5,\ 0,\ s_6,\ \gg\rangle$
	$\langle s_2,\ 0,\ s_3,\ 1\rangle$	$\langle s_4,\ A,\ s_0,\ \gg\rangle$	

End.



Enumerating TMs

Enumeration Theorem:

All TMs can be listed T_1, T_2, ..., T_n, ... in such a way that each index n completely determines the corresponding TM.

- *Idea*: A TM is completely determined by its set of transition rules.
- *So*: A TM corresponds to a (perhaps very long) string of symbols drawn from $\{q_0, q_1, ..., q_n\}$ and $\{s_0, ..., s_m\}$.

Successor function TM $= s_0 1 s_0 \gg s_0 0 s_1 1 s_1 1 s_1 \ll s_1 0 s_2 \gg$

Copier TM $= s_0 1 s_0 A s_0 A s_1 \gg s_0 0 s_5 \ll s_1 1 s_1 \gg s_1 0 s_2 \gg s_2 1 s_2 \gg s_2 0 s_3 1 s_3 1 s_3 \ll s_3 0 s_4 \ll s_4 1 s_4 \ll$
$s_4 A s_0 \gg s_5 A s_5 1 s_5 1 s_5 \ll s_5 0 s_6 \gg$

- *Now*: Encode these symbols as natural numbers...

One way to do this:

symbol	*code# for symbol*
$\gg$	3
$\ll$	5
s_i	$7 + 4i$
q_i	$9 + 4i$

code# for symbol strings

For symbol string $u_1 ... u_j$ that represents Turing machine T:

$$code\#(T) = p_1^{code\#(u_1)} \times ... \times p_j^{code\#(u_j)}$$

where $p_1, p_2, ..., p_j$ are the first j prime numbers 2, 3, 5, ...

- *So*: Each TM T_n corresponds to exactly one natural number $code\#(T_n)$.
- *And*: Any natural number can be decoded (by its unique prime factorization) to determine if it corresponds to a TM.

2. The Halting Problem

- Is there a TM that can determine whether or not any given TM T_t halts?
- _Or_: Is there a TM that can compute the *halting function* $h(t, n)$?

Halting function $h(t, n)$

$$h(t, n) = \begin{cases} 0 \text{ if } T_t \text{ halts on input } n. \\ 1 \text{ if } T_t \text{ does not halt on input } n. \end{cases}$$

*Claim*: $h(t, n)$ is not *Turing-computable* (*i.e.*, no TM can compute it).

- _*Proof*_: Suppose there's a TM, H, that computes $h(t, n)$.

  This means:

Halting TM, H

$$\text{On input } n, t, \begin{cases} H \text{ halts with output 0 if } T_t \text{ halts on input } n. \\ H \text{ halts with output 1 if } T_t \text{ does not halt on input } n. \end{cases}$$

Now show that H cannot exist.

Step 1: Construct another TM, H', that computes $h(n, n)$.

- *This can be done by attaching the copier TM to the front of H.*

$H' = H + copier$

On input n, $\begin{cases} H' \text{ halts with output 0 if } T_n \text{ halts on input } n. \\ H' \text{ halts with output 1 if } T_n \text{ does not halt on input } n. \end{cases}$

Step 2: Construct a "loop" TM which does the following:

loop

$\begin{cases} \text{On input 0, } \textit{loop} \text{ does not halt.} \\ \text{On input 1, } \textit{loop halts}. \end{cases}$

Step 3: Now attach *loop* to the end of H' to produce a TM, M.

$M = loop + H + copier$

On input n, $\begin{cases} M \text{ does not halt if } T_n \text{ halts on input } n. \\ M \text{ halts if } T_n \text{ does not halt on input } n. \end{cases}$

- This says that M halts *if and only if* T_n does not halt.

Now: Suppose M occurs as T_{n_0} in the list of all TMs.

- What happens when we feed M its own code number n_0 as input?

> *M = copier + H + loop, given input n_0*
>
> $$\text{On input } n_0, \begin{cases} M \text{ does not halt if } T_{n_0} \text{ halts on input } n_0. \\ M \text{ halts if } T_{n_0} \text{ does not halt on input } n_0. \end{cases}$$

- This says that M halts *if and only if* M does not halt!

There can be no such M!

- Since the *copier* and *loop* TMs are possible, this must mean there can be no Halting TM, H.
- So the Halting function is not Turing-computable.

Why should this matter?

2. Classical (Turing) Computability

- *What does it mean to say something is computable?*
  - Suppose the somethings of interest are functions on the natural numbers $\mathbb{N}$.
  - To say a function on $\mathbb{N}$ is computable is (in some sense) to say that there's an "algorithm" which, if followed by a computer would calculate the value of that function, given the appropriate type of input.
  - Can this be made more precise?

Turing Thesis:

A (partial) function on $\mathbb{N}$ is computable by algorithm *if and only if* it is Turing computable.

- *In other words*: Turing machines provide us with a precise notion of computability... (for computing functions on $\mathbb{N}$).

(a) Why accept Turing's Thesis?

Church's Thesis:

A (partial) function on $\mathbb{N}$ is computable by algorithm *if and only if* it is a recursive partial function.

Alonzo Church
(1903-1995)

- *Idea*: The computable functions are those that can be recursively generated from a small set of axioms (this can be made mathematically precise).
- *Key result*: A partial function on $\mathbb{N}$ is Turing computable *if and only if* it is a partial recursive function. (So Turing's Thesis is equivalent to Church's Thesis.)
- *Moreover*: Other models of computability (abacus machines, *etc.*) can be shown to be equivalent to Turing computability.

But I want to compute functions on the real numbers $\mathbb{R}$, not just $\mathbb{N}$!

mathematical physicist

logician

Let me work on it...

(b) The Limits of Turing Computability

Def. A problem is *Turing solvable* if there's a TM that can solve the problem after a finite number of steps.

<u>*Turing unsolvable problems*</u>:

(i) *The halting problem*. Problem of deciding, given an arbitrary TM, whether or not it will halt.

(ii) *The decision problem for 1st-order logic*. Problem of deciding the validity or invalidity of an arbitrary sentence of 1st-order logic.

- *There's a TM that will halt after finite steps with output "Yes" for any valid 1st-order sentence as input; but there's no TM that will halt after finite steps with output "Yes" for any invalid 1st-order sentence as input.*
- *A "Yes" TM for validity is not the same as a "Yes" TM for invalidity!*

(iii) *The decision problem for 1st-order arithmetic*. Problem of deciding the validity or invalidity of an arbitrary sentence of 1st-order arithmetic.

- *There's no "Yes" TM for validity and no "Yes" TM for invalidity for 1st-order arithmetic (one consequence of Gödel's Incompleteness Theorem).*

- Is Fermat's Last "Theorem" really a theorem?

For $n \geq 3$, there are no whole numbers x, y, z such that $x^n + y^n = z^n$.

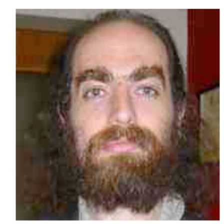
Pierre de Fermat
(1607-1665)

Proven by Andrew Wiles in 1993 after 3 centuries of work.

- Is the Poincaré Conjecture a theorem?

Every simply connected closed 3-manifold is homomorphic to the 3-sphere. (Or: the 3-sphere is the only type of bounded 3-dim space that contains no holes.)

Henri Poincaré
(1854-1912)



Proven by Grigori Perelman in 2003 after a century and $1million prize (declined!).

- Wouldn't it be easier if there were a program that decided which statements were theorems and which weren't?
- *But*: No TM (hence classical computer) can in principle tell us!