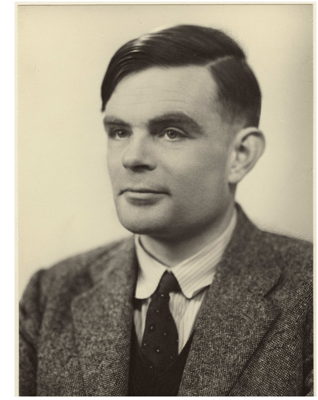
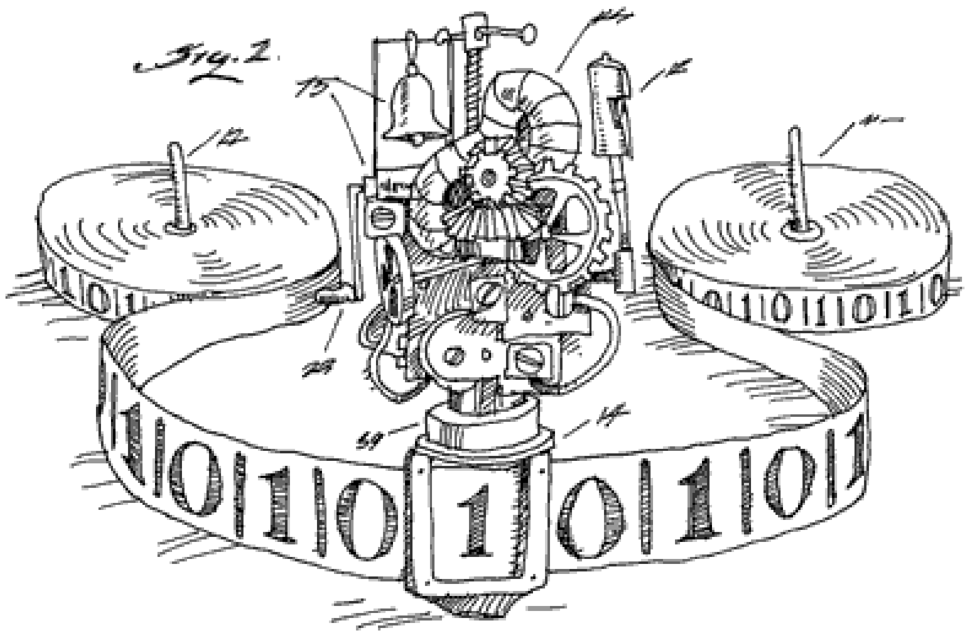
Foundations of Computer Science
Lecture 26

Turing Machines

- The Turing Machine: DFA with Random Access Memory (RAM)
- Transducer Turing Machines
- Infinite Loops
- Encodings of Turing Machines

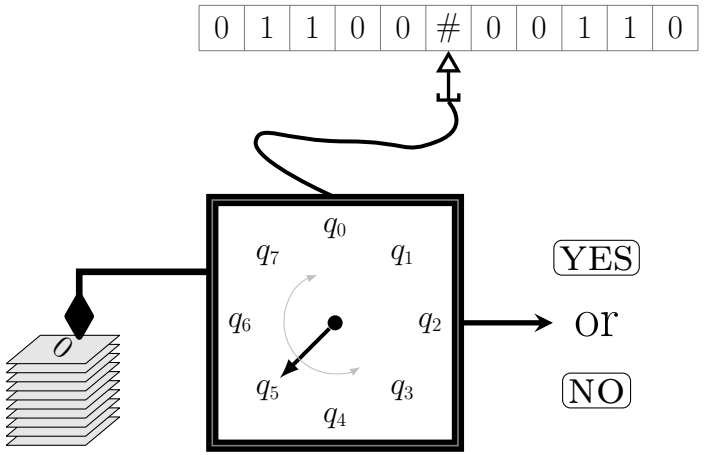




Last Time: CFGs and Pushdown Automata

$$\mathcal{L} = \{w\#w^{\mathrm{R}} \mid w \in \{0,1\}^*\}$$

$$S \to \# \mid 0S0 \mid 1S1$$



DFA with stack memory (push, pop, read).

Push the first half of the string.

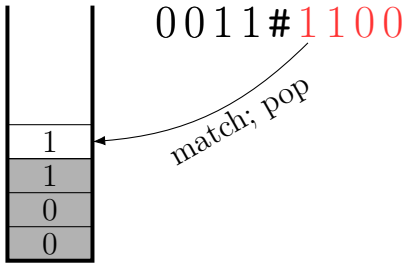
For each bit in the second half, pop the stack and compare.

DFAs with stack memory closely related to CFGs.

Non Context Free

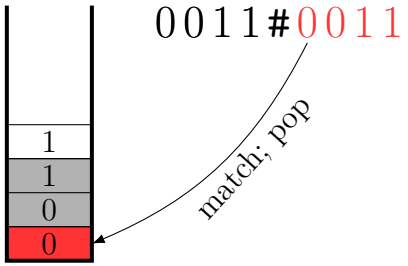
$\{w\#w\}$	repetition
$\{0^{\bullet n}1^{\bullet n}0^{\bullet n}\}$	multiple-equality
$\{0^{\bullet n^2}\}, \{0^{\bullet n}1^{\bullet n^2}\}$	squaring
$\{0^{\bullet 2^n}\}, \{0^{\bullet n}1^{\bullet 2^n}\}$	exponentiation

$\underline{w\#w^{\mathrm{R}}}$



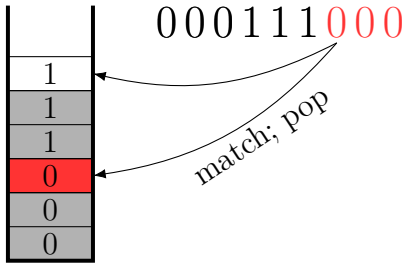
0011 is pushed.
DFA matches 1100 by popping.

$\underline{w\#w}$



0011 is pushed.
DFA needs bottom-access to match.

$\underline{0^{\bullet n}1^{\bullet n}0^{\bullet n}}$



000111 is pushed onto the stack.
DFA needs random access to match.

The file clerk who only has access to the top of his *stack* of papers has fundamentally less power than the file clerk who has a *filing cabinet* with access to all his papers.

We need a new model, one with Random Access Memory (RAM).

Today: Turing Machines

1. Solving a non context free language: $w\#w^{\mathrm{R}}$.
2. Transducer Turing Machines.
3. Infinite Loops
4. Encodings of Turing Machines

Turing's 1936 Miracle

"On Computable Numbers with an Application to the Entscheidungsproblem"

A classic which epitomizes the beauty of pure thought, where Alan Turing

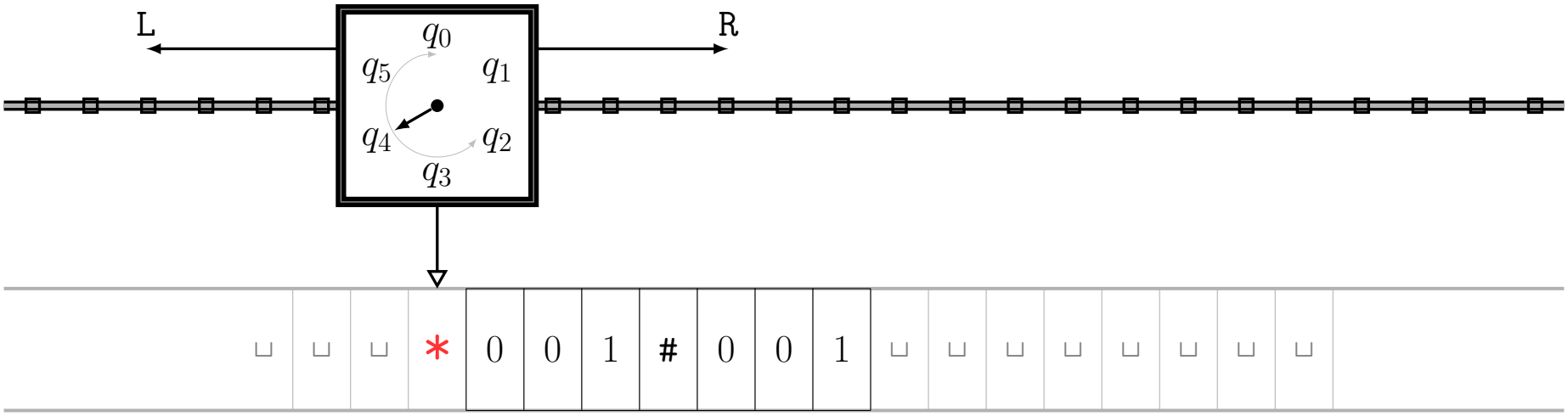
- Invented a notion of what it means for a number to be computable.
- Invented the computer.
- Invented and used subroutines.
- Invented the *programmable* computer.
- Gave a negative answer to Hilbert's Entscheidungsproblem.

All this before the world even saw its first computer. Wow!

(Oh, and by the way, he helped Britain win WWII against Hitler by decrypting the Enigma machine.)

. . . and for all this, society drove him to suicide. Go figure!

Turing's Machine



- States.
- Can move L/R (or stay put) giving random access to an infinite read-write tape.
- Input written on the tape to start.
- Instructions specify what to do based on state and what is on the tape.
- Beacon symbol $*$ (start of the tape).

Let's see the capabilities of this machine on the non context free problem

$$\mathcal{L} = \{w\#w\}.$$

Solving $w\#w$

001#001

1: Check for one "#", otherwise REJECT (a DFA can do this).

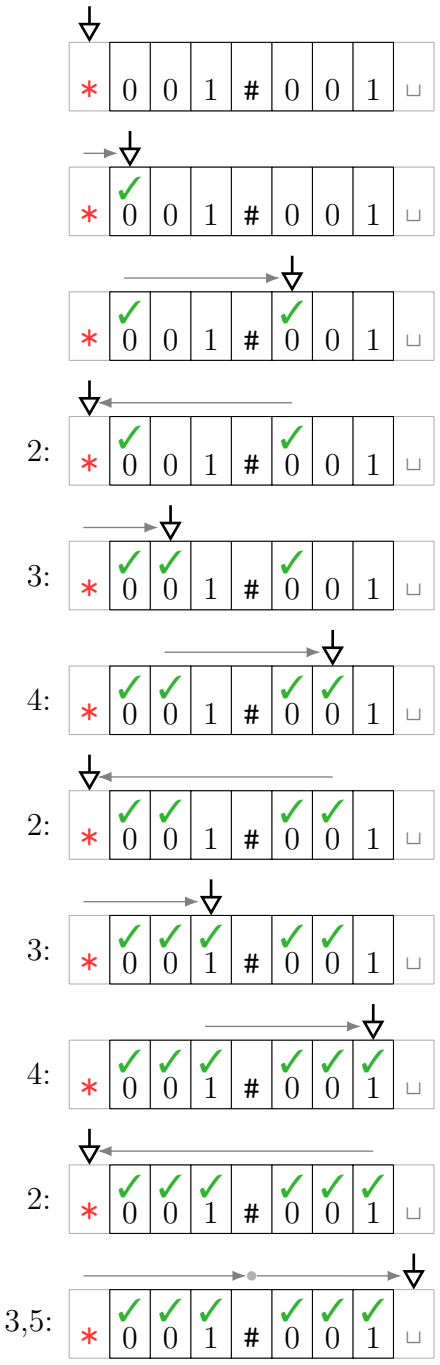
2: Return to "$*$".

3: Move right to first non-marked bit *before* "#".
Mark the location and *remember* the bit.
(If you reach "#" before any non-marked bit, GOTO step 5.)

4: Move right to first non-marked bit *after* "#".
If you reach "␣" before any non-marked bit, REJECT.
If the bit does not match the bit from step 3, REJECT.
Otherwise (bit matches), mark the location. GOTO step 2.

5: Move right to first non-marked bit *after* "#".
If you reach "␣" before any non-marked bit, ACCEPT.
If you find a bit (string on the right is too long), REJECT.

YES

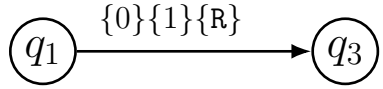


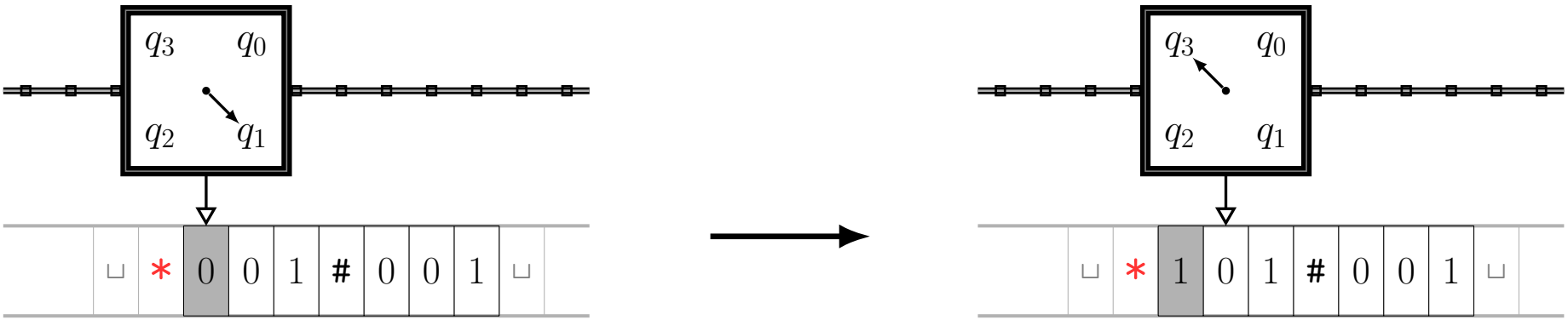
Turing Machine Instructions

DFA instruction: $q_1 0 q_3$ ← if in q_1 and read 0, transition to q_3

$$\{q_1\}\ \{0\} \rightarrow \{q_3\}\ \{1\}\ \{\mathtt{R}\}$$

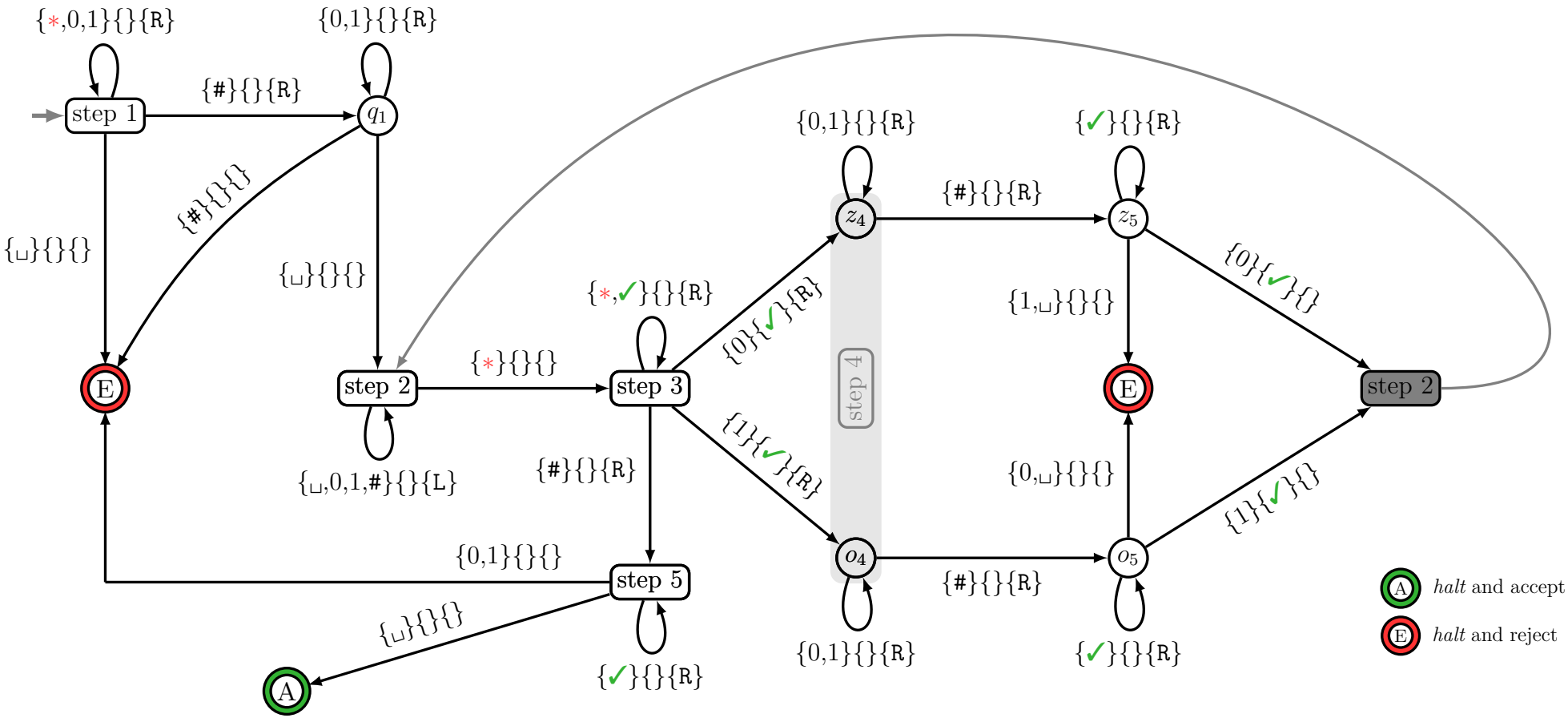
current state read next state write move





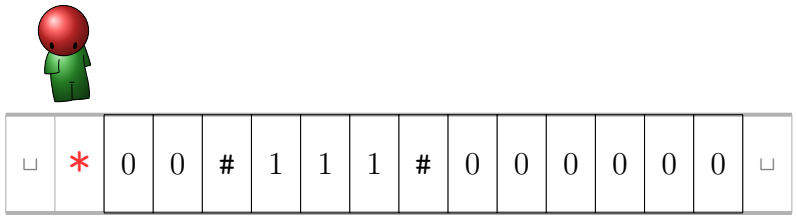
Building the Turing Machine that Solves $w\#w$

1. Check for one "#", otherwise REJECT (a DFA can do this).
2. Return to "$*$".
3. Move right to first non-marked bit *before* "#".
   Mark and *remember* the bit. If you reached "#" before any non-marked bit, GOTO step 5.
4. Move right to first non-marked bit *after* "#".
   If you reach "$\sqcup$" or bit does not match, REJECT. If bit matches, mark the location. GOTO step 2.
5. Move right to first non-marked bit *after* "#".
   If you reach "$\sqcup$", ACCEPT. If you come to a non-marked bit, REJECT.



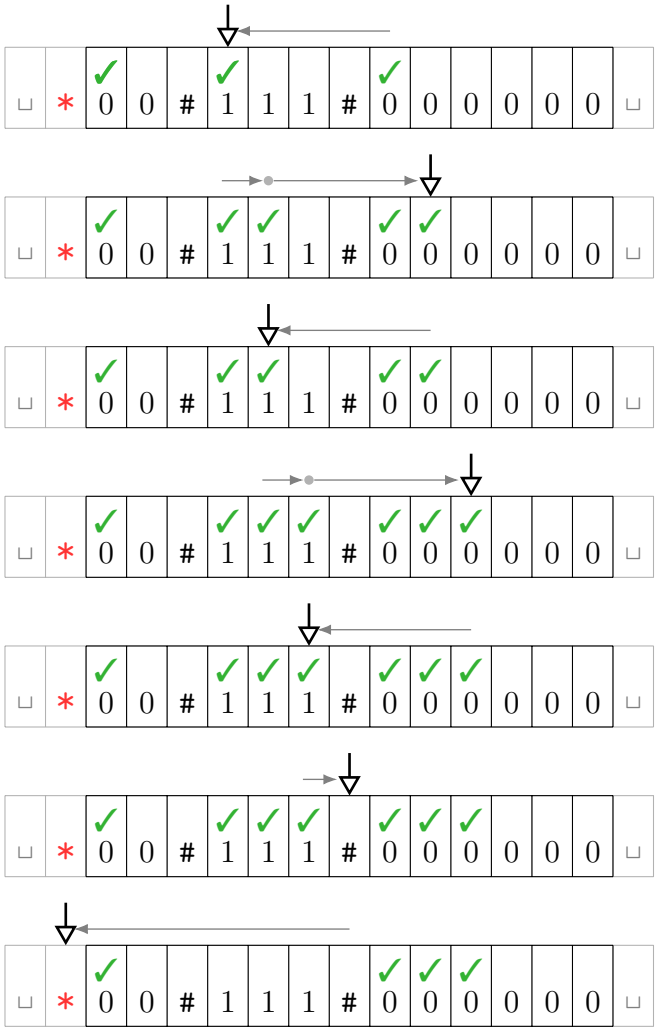
Turing Machine for Multiplication

$$\mathcal{L}_{\text{mult}} = \{0^{\bullet i}\#1^{\bullet j}\#0^{\bullet k} \mid i, j > 0 \text{ and } k = i \times j\}$$

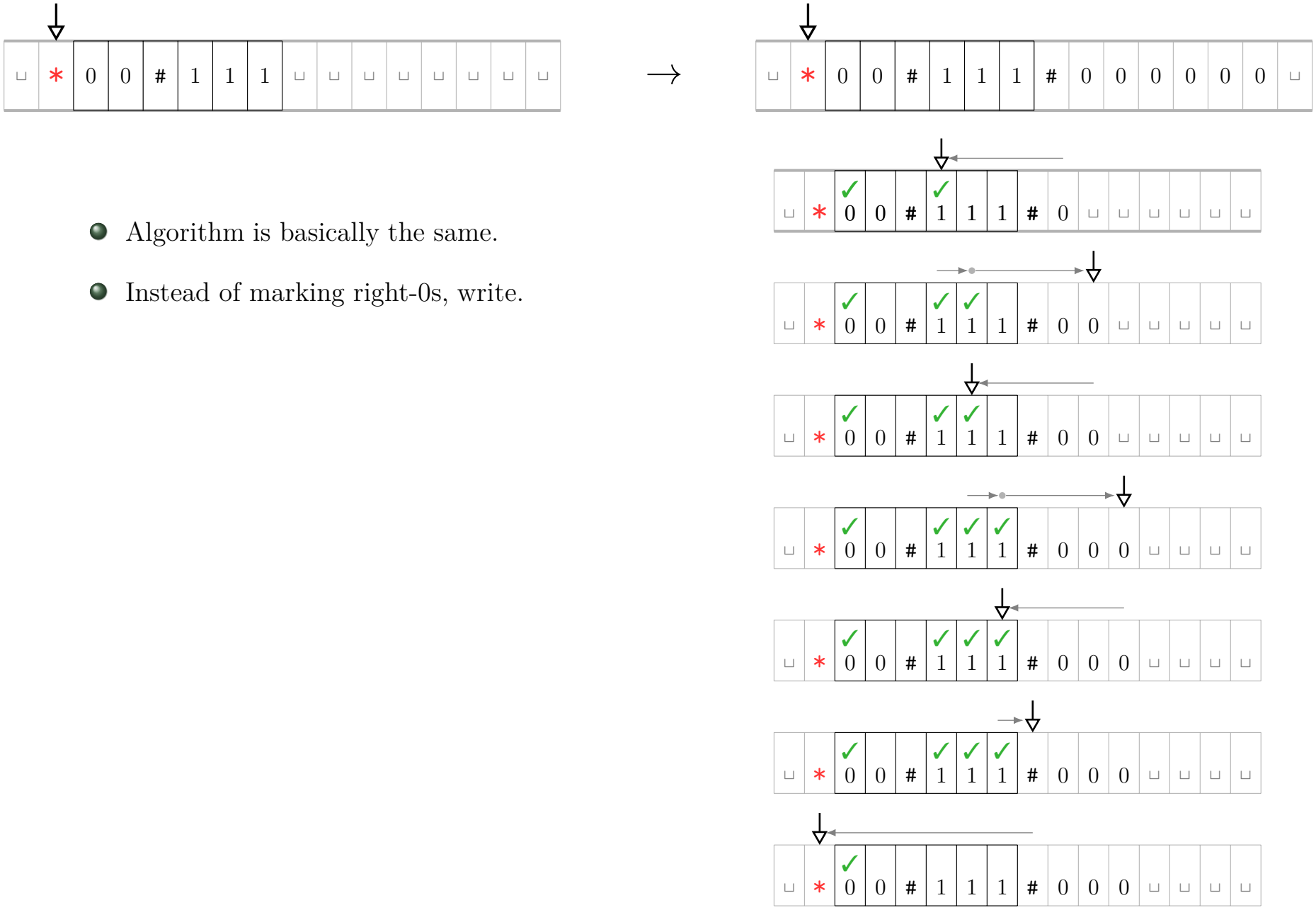


Multiplication is repeated addition.
Pair each left-0 with a block of right-0s equal to the number of 1s

1: Verify the input format is $0^{\bullet i}\#1^{\bullet j}\#0^{\bullet k}$ (A DFA can solve this).
2: Return to $*$.
3: Move right to mark first unmarked left-0, then right to "#".
   If no unmarked left-0's (you reach "#"), GOTO step 6.
4: Move right and mark first unmarked 1.
   If all 1's marked (reach "#") move left, unmarking 1's. GOTO step 2.
5: Move right to find an unmarked right-0.
   If no unmarked right-0's (come to "␣"), REJECT
   Otherwise, mark the 0, move left to first marked 1. GOTO step 4.
6: Move right to verify there are no unmarked right-zeros.
   If come to unmarked right-zero, REJECT; if come to "␣" ACCEPT.

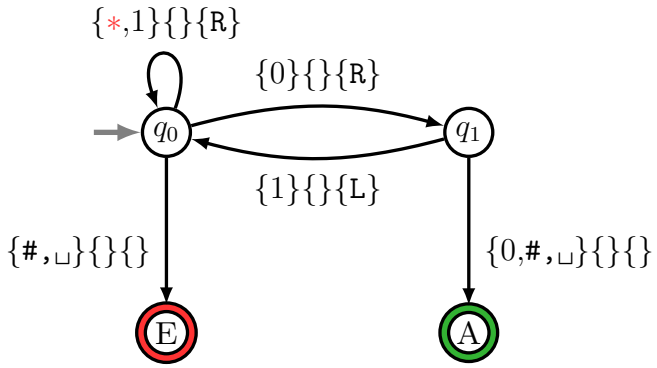


Transducer Turing Machine That Multiplies



- Algorithm is basically the same.
- Instead of marking right-0s, write.

Infinite Loops



$$M(w) = \begin{cases} \text{Halts in an accept state} & \rightarrow \textsc{accept} \\ \text{Halts in a reject state} & \rightarrow \textsc{reject} \\ \text{Loops forever} & \rightarrow ? \end{cases}$$

What happens if the input is 01?

Turing Machine M is a *recognizer* for language $\mathcal{L}(M)$:

$$w \in \mathcal{L}(M) \leftrightarrow M(w) = \text{halt with a } \boxed{\text{YES}};$$
$$w \notin \mathcal{L}(M) \leftrightarrow M(w) = \text{halt with a } \boxed{\text{NO}} \textit{ or loop forever}.$$

Turing Machine M is a *decider* for language $\mathcal{L}(M)$:

$$w \in \mathcal{L}(M) \leftrightarrow M(w) = \text{halt with a } \boxed{\text{YES}};$$
$$w \notin \mathcal{L}(M) \leftrightarrow M(w) = \text{halt with a } \boxed{\text{NO}}.$$

Practical *algorithms* must halt! Practical algorithms correspond to deciders.

Encoding a Turing Machine as A Bit-String

Mathematical Description of a Turing Machine

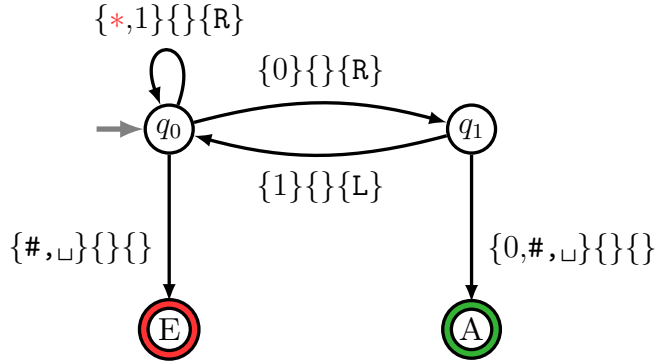
1. **States Q.** The first state is the start state, the halting states are A,R.
2. **Symbols Σ.** By default these are {$*$, 0, 1, ␣, #}.
3. **Machine-level transition instructions.** Each instruction has the form

   {state}{read-symbol}{next-state}{written-symbol}{move}

   The instructions map each (state,symbol) pair to a (state,symbol,move) triple and thus form a *transition function* $\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{\mathtt{L},\mathtt{R},\mathtt{S}\}$.

1. **States.** $\{q_0, q_1, \mathrm{A}, \mathrm{E}\}$
2. **Symbols.** {$*$, 0, 1, ␣, #}
3. **Machine-level transition instructions.**

$\{q_0\}\{*\}\{q_0\}\{*\}\{\mathtt{R}\}$
$\{q_0\}\{1\}\{q_0\}\{1\}\{\mathtt{R}\}$
$\{q_0\}\{0\}\{q_1\}\{0\}\{\mathtt{R}\}$
$\{q_0\}\{\#\}\{\mathrm{E}\}\ \{\#\}\{\mathtt{S}\}$
$\{q_0\}\{␣\}\{\mathrm{E}\}\ \{␣\}\{\mathtt{S}\}$
$\{q_1\}\{1\}\{q_0\}\{1\}\{\mathtt{L}\}$
$\{q_1\}\{0\}\{\mathrm{A}\}\ \{0\}\{\mathtt{S}\}$
$\{q_1\}\{\#\}\{\mathrm{A}\}\ \{\#\}\{\mathtt{S}\}$
$\{q_1\}\{␣\}\{\mathrm{A}\}\ \{␣\}\{\mathtt{S}\}$



The description of a Turing Machine is a *finite* binary string.
Turing machines are countable and can be listed: $\{M_1, M_2, \ldots\}$.
The problems solvable by an algorithm are countable: $\{\mathcal{L}(M_1), \mathcal{L}(M_2), \ldots\}$.