Models of computation

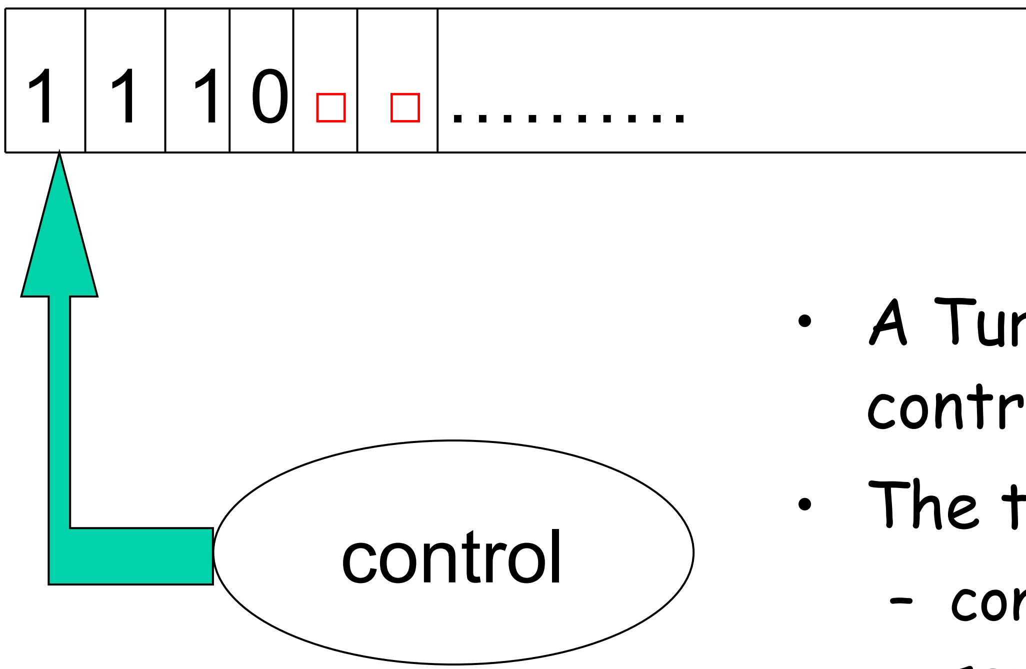
The following models attempt to model computers running certain programs/algorithms.

- Finite State Automata: no memory
- Pushdown Automata: memory in the form of a stack
- Turing Machines: arbitrarily read/write memory

Too restricted

A Turing machine can model any computing devices running a particular algorithm/program.

Turing machines (TM)



Read-write tape

NB. □ = blank symbol

- A Turing machine = a finite state control + a read-write tape.
- The tape
  - contains the input initially, also serves as the working memory.
  - Each unit (square, cell) on the tape can host a symbol.
  - The tape is **infinite** to the right.

In one step

There is a read-write head pointing to a square on the tape.

In one step (move),

- the control reads the symbol pointed by the read-write head;
- then depending on the current state and the tape symbol, the control
  - changes its state,
  - (over)writes on the current tape square, and
  - moves the tape head to the left or right.

Accept or Reject

A Turing machine always starts from a particular state, called the start state.

Given an input string w on the tape, a Turing machine operates step by step until it reaches one of the following two special states, then it halts.

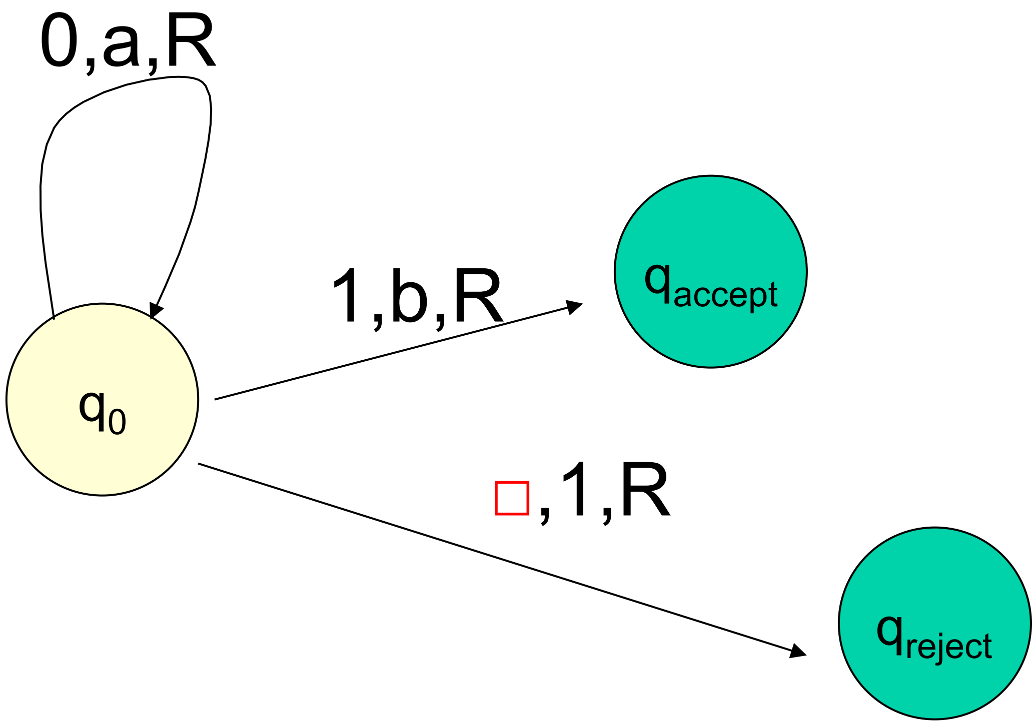
- accepting (YES) state: The input w is said to be accepted.
- rejecting (NO) state: The input w is said to be rejected.

NB. By definition, a Turing machine may accept/reject without reading all input symbols on the tape.

Example

Consider a Turing machine M with three states:

q_0 (the start state), q_{accept}, q_{reject}



Tape

00001111 □ □ □ …

□ = blank symbol

NB. The label "0,a,R" means that if symbol 0 is read, the machine overwrites the 0 with a and moves the tape head to the **r**ight.

Definition

A Turing machine is a 7-tuple, (Q, Σ, Γ, δ, q_0, q_{accept}, q_{reject}), where

- Q is a finite set of states (including q_0, q_{accept}, q_{reject});
- Σ and Γ are finite set of symbols; Σ is called the input alphabet and Γ is called the tape alphabet;
- $\Sigma \subset \Gamma$.

  Γ, but not Σ, contains a special symbol called blank symbol, denoted $\square$;
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ is the transition function. E.g., $\delta(q, a) = (q', b, R)$

NB. $\square$ is not in Σ.

Transition function

$\delta : Q - \{q_{accept}, q_{reject}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$

δ specifies the behavior of the TM.

Precisely, it specifies how the TM operates for every possible combination of state and tape symbol.

For example, $\delta(q, a) = (q', b, R)$ means that when the current state is q and the symbol currently under the tape head is a, the machine will

- write the symbol b on the tape replacing a,
- move the tape head to the right by one square, and
- go to the state q'.

Configurations

At any time, the **configuration** of a TM refers to a complete description of the machine, comprising

- the current state;
- the position of the tape head;
- the content of the tape.

Notation: *uqv*, where u and v are strings in Γ^*.

- the current state = q
- the tape content = uv followed by all blank symbols
- tape head position = at the first symbol of v.

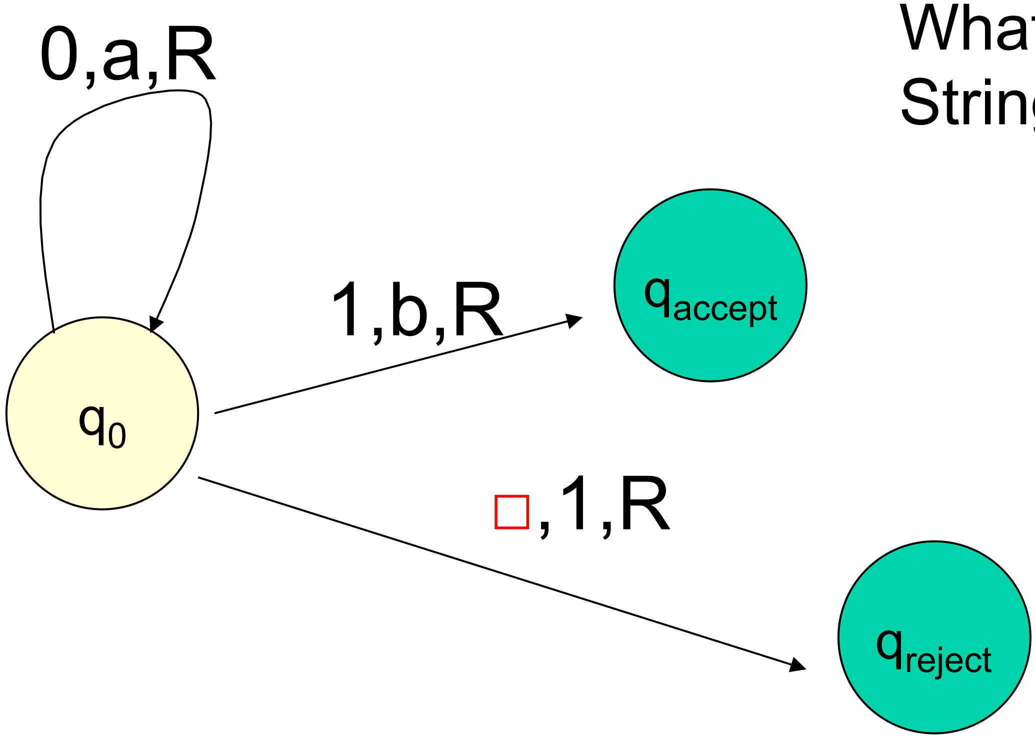
Given an input w, the initial/start configuration is:

- q_0w

Example

Consider a Turing machine M with three states:

q_0 (the start state), q_{accept}, q_{reject}



What inputs does M accept?
Strings containing a "1".

00001111 □ □ □ …

□ = blank symbol

Examples

- Suppose the current configuration is $0001q000$ and $\delta(q, 0) = (q', 1, R)$.
  The next configuration is $00011q'00$.

- Suppose the current configuration is $0001q000$ and $\delta(q, 0) = (q', 0, L)$.
  The next configuration is $000q'1000$

- Suppose the current configuration is $0001q$ and $\delta(q, \square) = (q'', 1, L)$.
  The next configuration is $000q''11$

Notation

A configuration C1 is said to yield another configuration C2, denoted by **$C1 \Rightarrow C2$**,
if the Turing machine can move from C1 to C2 in one step.

$C1 \overset{*}{\Rightarrow} C2$ if the TM can move from C1 to C2 in zero or more steps.

Boundary case

What happens when a TM attempts to move beyond the left end of the tape?

- The tape head simply stays at the leftmost square.
- That is, suppose the current configuration is qav,

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

  then the next configuration is $q'bv$.

Halting configurations: current state = q_{accept} or = q_{reject}

- The machine stops (next move is undefined).

Question: Given an input w, does a TM always halt ?

Halting configurations

Two possible halting configurations:

Accepting configuration: current state = q_{accept}

Rejecting configuration: current state = q_{reject}

A Turing machine M accepts an input w if M, starting with configuration q_0w, can eventually arrive at an accepting configuration.

A Turing machine M rejects an input w if M, starting with q_0w, can eventually arrive at a rejecting configuration.

- Question: Given an input w, does a TM always halt at an accepting/rejection configuration.

Halting configurations

Two possible halting configurations:

Accepting configuration: current state = q_{accept}

Rejecting configuration: current state = q_{reject}

A Turing machine M accepts an input w if M, starting with configuration q_0w, can eventually arrive at an accepting configuration.

A Turing machine M rejects an input w if M, starting with q_0w, can eventually arrive at a rejecting configuration.

- Question: Given an input w, does a TM always halt at an accepting/rejection configuration. No.

High level description

The transition function (state transition diagram) of a TM can be very tedious for non-trivial problems.

Usually, I just put down a high level description. You need to draw the state transition diagram yourself.

Example 1: Let Σ = {a, b, c}. A string $x \in \Sigma^*$ is said to be a palindrome if x reads the same backwards.

- E.g., "abcba", "abccccba" are palindromes, but "abca" isn't.

Design a TM to determine whether a given string x is a palindrome.

Palindrome

a b c a a c b a □ □ □ □....

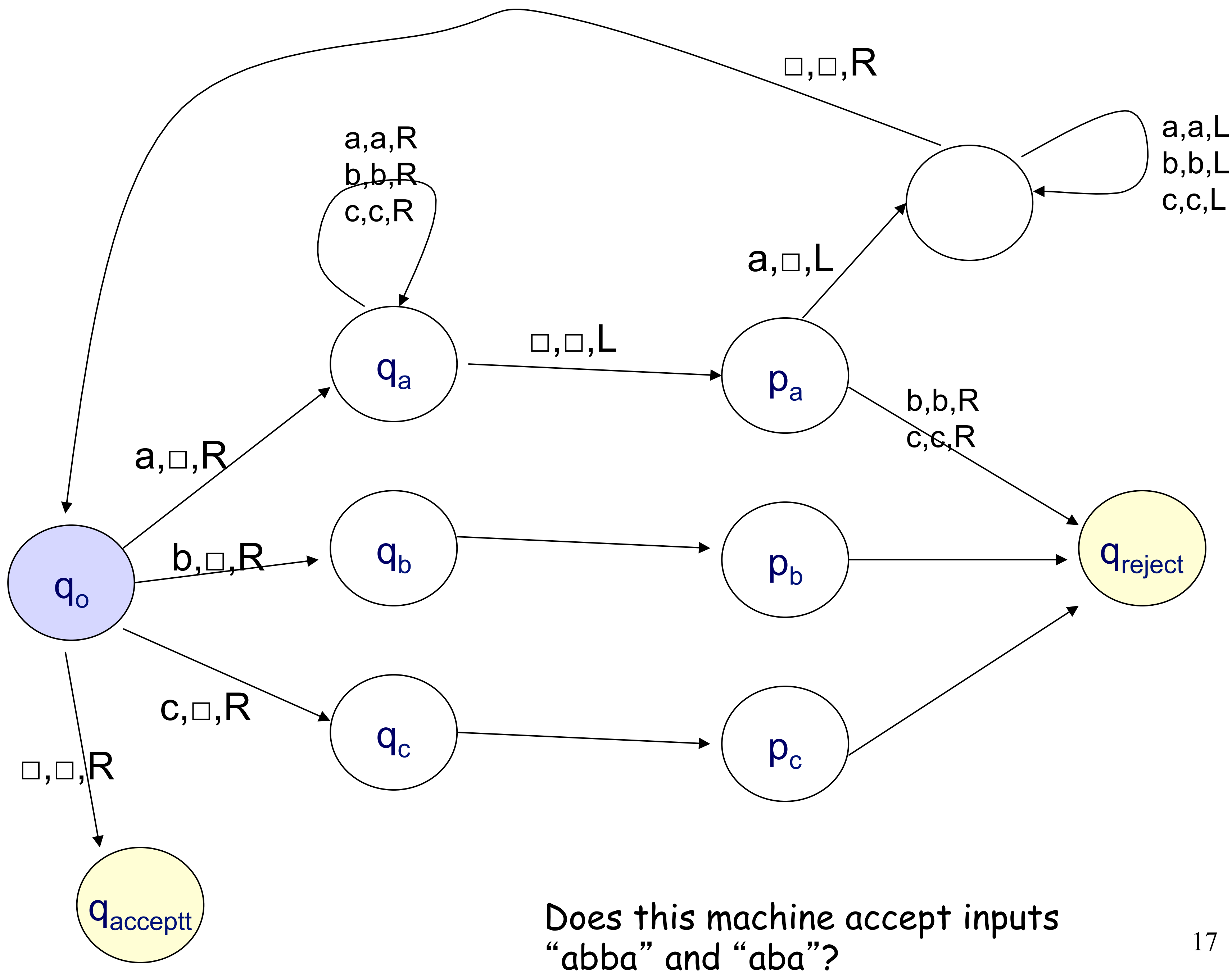
↑

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

- Read the next symbol.
- If it is a blank symbol, go to q_{accept}.
- Otherwise remember the symbol (how? use 3 different states q_a, q_b, q_c) and replace the symbol with □.
- If the next symbol = □, go to q_{accept} ;

  Otherwise move the tape head to the last non-blank symbol, check whether this symbol matches the symbol memorized.
- If no match, go to q_{reject}.

  Otherwise replace the last symbol with □, move the tape head to the leftmost non-blank symbol, and repeat the steps above.



Does this machine accept inputs "abba" and "aba"?

More examples

Examples from Sipser's book.

3.7: $A = \{ 0^m \mid m = 2^n \text{ for some } n > 0\}$.

3.9: $B = \{ w\#w \mid w \in \{0,1\}^* \}$.

3.11: $C = \{a^i b^j c^k \mid i \times j = k \text{ and } i,j,k \geq 1\}$.

3.12: $E = \{ \# x_1 \# x_2 \# x_3 \# \ldots \# x_k \mid$ each x_i is a distinct binary number $\}$

Language acceptance

The language of a Turing machine M, denoted by L(M), is the set of strings M accepts.

For any input $w \in \Sigma^*$,

if $w \in L(M)$, M with input w will arrive at an accepting configuration.

However, if x is **NOT** in L(M), what do we know?

- M with x will not arrive at an accepting configuration. True or false?
- M with x will arrive at a rejecting configuration. True or false?

Decidable & recognizable languages

A language L is <u>Turing-recognizable</u> (Turing-acceptable, recursive enumerable) if there exists a Turing machine M such that L(M) = L.

A language L is <u>Turing-decidable</u> (decidable, recursive) if there exists a Turing machine T such that L(T) = L and T halts on all possible inputs.

I.e., for any input $w \in L$, T halts and accepts w; and
for any input $w \notin L$, T halts and rejects w.

NB. The machine M is said to <u>recognize</u> L, and the machine T is said to <u>decide</u> L.

True, False, ???

- If L is Turing-decidable, L is Turing-recognizable.
- If L is Turing-recognizable, L is Turing-decidable.

True, False, ???

True
- If L is Turing-decidable, L is Turing-recognizable.

False
- If L is Turing-recognizable, L is Turing-decidable.

Decidability

Notation: For any dfa A, let $\langle A \rangle$ denote a binary string encoding A, let $\langle A,x \rangle$ denote a binary string encoding A and an input x.

Decision problem: Given a dfa A and an input x, determine A accepts x or not.

Language: $L = \{ \langle A,x \rangle \mid A \text{ is a dfa that accepts string } x\}$.

Decidability

Notation: For any dfa A, let $\langle A \rangle$ denote a <u>binary string</u> encoding M, let $\langle A,x \rangle$ denote a binary string encoding M and an input x.

Decision problem: Given a dfa A and an input x, determine A accepts x or not.

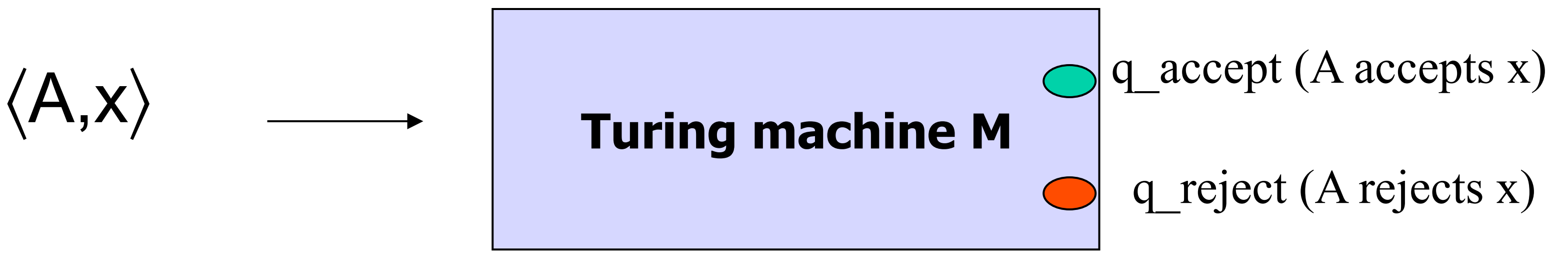
Language: $L = \{ \langle A,x \rangle \mid A$ is a dfa that accepts string $x\}$.

Can you write a C program to solve the above problem?

Turing machines

Is L decidable ?

Or equivalently, is there a Turing machine M that decides L (M must halt on any input and tell the membership correctly) ?



NB. To prove L is decidable, it is not sufficient to show that, for each ⟨A,x⟩, we can construct a Turing machine T to simulate A on x.
I.e., We need a Turing machineT to work for all ⟨A,x⟩.

Simulation

$\langle A,x \rangle$

…. (q,a,q') …q_0…. # $x_1x_2…x_n$ #

How can M simulate any given dfa A?

• Check whether $\langle A,x \rangle$ contains the 5 components of a dfa. If not, M rejects $\langle A,x \rangle$.

q_accept (A accepts x)

• Find the start state q_0 from $\langle A,x \rangle$ and copy it to the end of the tape; find the first input x_1 from $\langle A,x \rangle$ and replace it with \$.

$\langle A,x \rangle$

…. (q,a,q') …q_0…. # \$$x_2…x_n$ # **q_0, x_1**

Scan the transition function in $\langle A,x \rangle$ to find (q_0 , x_1,q). Then update q_0 with q. Find the next input x_2 …..

…. (q,a,q') …q_0…. # \$\$$x_3…x_n$ # **q, x_2**

Undecidability

Notation: For any TM M, let ⟨M⟩ denote a binary string encoding M, let ⟨M,x⟩ denote a binary string encoding M and an input x.

Decision problem: Given a Turing machine M and an input x, determine M accepts x or not.

Can you write a C program to solve the above problem?

Undecidable languages

Classical examples of languages that are not Turing-decidable (non-recursive).

- A_{TM} = { ⟨M,x⟩ | M is a TM and M accepts x }.
- K = { ⟨M⟩ | M is a TM and M accepts ⟨M⟩ }.

Denote ~K = { ⟨M⟩ | M is a TM and M doesn't accept ⟨M⟩ }.

Is K Turing recognizable?

Theorem: K is undecidable

Suppose, for the sake of contradiction, that K is decidable.
Then ~K is also decidable.

There exists a TM D deciding ~K.

By definition, for any <u>input</u> $\langle M \rangle$,

- D halts on <M>; and
- D accepts $\langle M \rangle$ if $\langle M \rangle \in$ ~K; and
- D rejects $\langle M \rangle$ if $\langle M \rangle \notin$ ~K

Contradiction

K = { ⟨M⟩ | M is a TM and M accepts ⟨M⟩ }.
~K = { ⟨M⟩ | M is a TM and M doesn't accept ⟨M⟩ }.
D decides ~K.

What happens if **D runs with the input ⟨D⟩**? Note that D must halt for all possible inputs, including ⟨D⟩.

- D accepts ⟨D⟩:
  - By definition of K, ⟨D⟩ ∈ K, or equivalently, ⟨D⟩ ∉ ~K.
  - As D decides ~K, D should reject ⟨D⟩.
- D rejects ⟨D⟩:
  - By definition of D, ⟨D⟩ ∉ ~K.
  - That means, ⟨D⟩ ∈ K; by definition of K, D accepts ⟨D⟩.

In both cases we obtain a contradiction. Thus, ~K is not decidable. Also, K is not decidable.

A_{TM} is not decidable.

Suppose, for the sake of contradiction, that A_{TM} is decidable. Then there exists a TM H deciding A_{TM}.

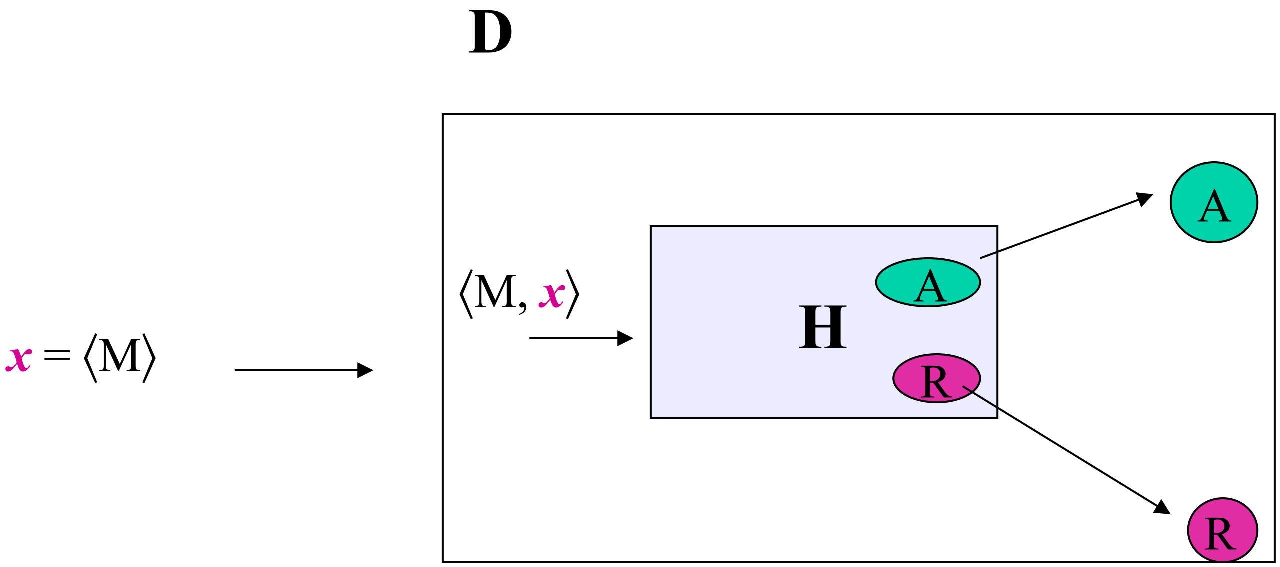
Below we show that H can be used to construct a TM D to decide K.

What is given? TM H decides A_{TM}. To construct: TM D for K.

Input to D: ***x*** = ⟨M⟩ for some TM M. (To ensure x is in proper format, D should first check whether ***x*** is a correct encoding of a TM first. If not, D rejects ***x*** immediately.)

D simulates H with input ⟨M, ***x***⟩. Note that H must halt. If H accepts (resp. rejects) then D accepts (resp. rejects).

Details of construction


D
$x = \langle M \rangle$
$\langle M, x \rangle$
H
A
R
A
R

Correctness of construction

Does D decide K?

For any TM M, let $x = \langle M \rangle$.

$x \in K \Rightarrow$ M accepts $x = \langle M \rangle$ (by def of K)

$\Rightarrow \langle M, x \rangle \in A_{TM}$

$\Rightarrow$ H accepts $\langle M, x \rangle$

$\Rightarrow$ D accepts x

$x \notin K \Rightarrow$ M doesn't accept $x = \langle M \rangle$ (by def of K)

$\Rightarrow \langle M, x \rangle \notin A_{TM}$

$\Rightarrow$ H rejects $\langle M, x \rangle$

$\Rightarrow$ D rejects x

D halts on all inputs and decides K, contradicting the fact that K is undecidable. Thus, A_{TM} is undecidable.

Questions

- Is K Turing-recognizable? Yes.

Construct a TM T that, given an input $\langle M \rangle$, simulates the machine M step by step.

If $\langle M \rangle \in K$, the simulation will stop eventually and T will accept. Therefore, L(T) = K.

- Is ~K Turing-recognizable? No.

If both K and ~K are Turing-recognizable, then K is decidable. A contradiction occurs.

Parallel simulation

- If L and ~L are Turing-recognizable, L is Turing-decidable.

Parallel simulation

- If L and ~L are Turing-recognizable, L is Turing-decidable.