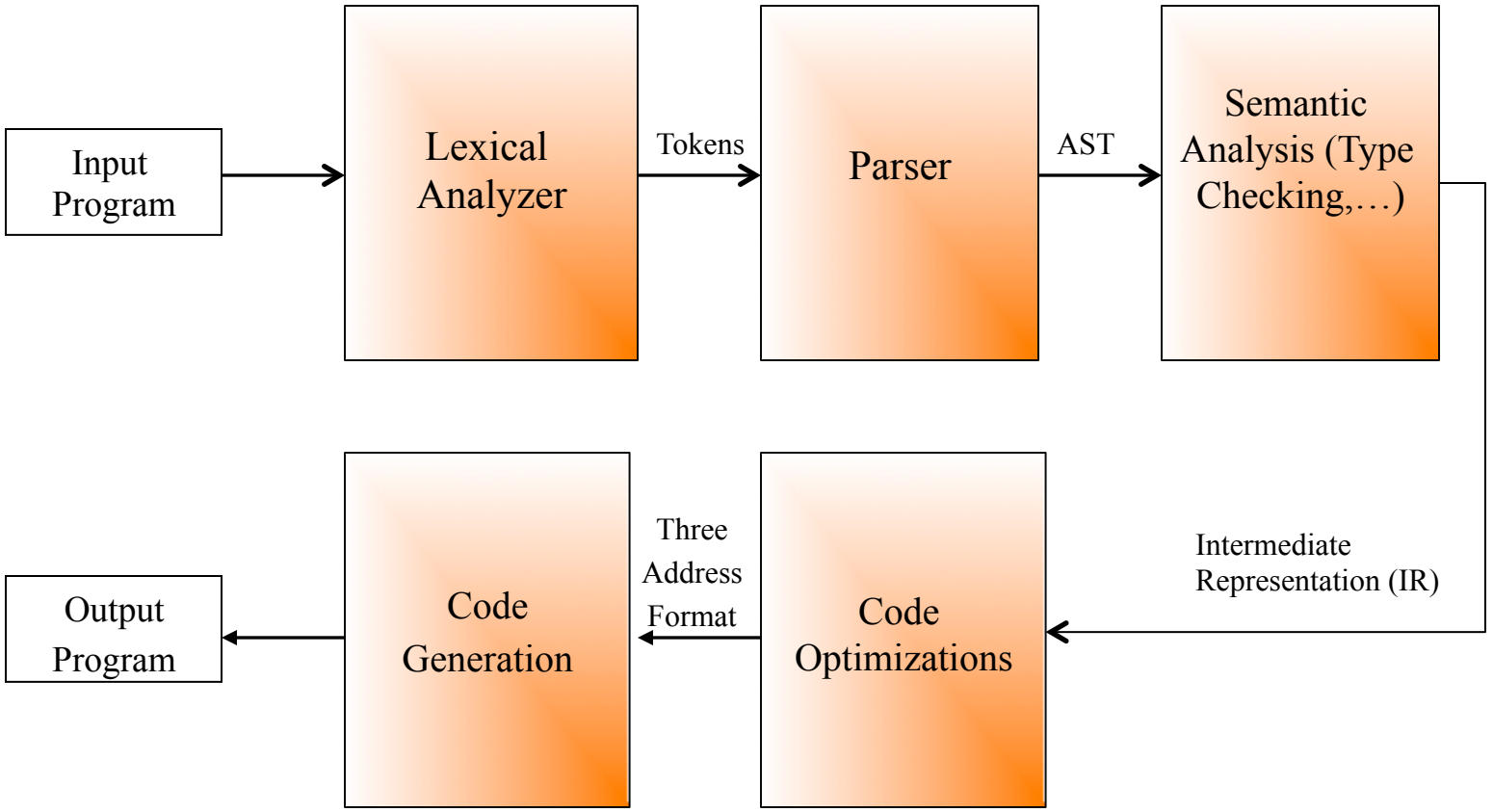
Basics of
Formal Languages and Computability

By
Vijay Ganesh

Last Class: The Structure (Stages/Phases) of a Compiler



Today's Lecture

- Formal languages
- Computability theory basics
- Concept of Undecidability
- Halting problem
- Undecidability of program analysis

Formal Languages

Language: a possibly infinite set of finite-length strings over a finite alphabet

String: a sequence (concatenation) of symbols from the alphabet of the language

Alphabet: Finite set of symbols/characters, e.g.,
$\Sigma = \{a,b,\ldots,z\}$ or $\{0,1\}$

Example:

Alphabet: $\Sigma = \{a,b\}$

Strings: ε, a, b, aaa, ababa, abbb,....

We represent the set of all strings over Σ as Σ^*

Language: A subset of strings of Σ^*

More Examples of Formal Languages

- The language over unary alphabet {a}: {ε, a, aa, aaa, …}
- Finite Languages: The cardinality of such language is a finite number, e.g., The set of all numbers less than 100
- Most languages we study have infinite cardinality: e.g., the set of even numbers
- We will study classes of formal languages such as regular, context-free and context-sensitive languages that are crucial for understanding compiler construction

Typical Operations on Strings

Consider a binary alphabet $\Sigma = \{a,b\}$

Operation name	Properties	Example
Concatenation(s1,s2)	Non-commutative, associative	$a.b \neq b.a$ $a.(b.c) = (a.b).c$ $\varepsilon.a = a.\ \varepsilon = a$
Prefix(s1) (Similarly define suffix)	Returns the prefix	ab is a prefix of abba ba is a suffix of abba
Reverse(s1)	Reverse(Reverse(S)) = S	Reverse of ab is ba
Length(s1)	Computes the number of chars in a string	Length(ab) =2 Length(ε) = 0

Typical Operations on Languages

Operation name	Properties	Representation
Union(L1, L2)	Set union of the corresponding sets of strings	$L1 \cup L2$
Intersection(L1, L2)	Set intersection of the corresponding sets of strings	$L1 \quad L2$
Difference(L1,L2)	Set difference of the corresponding sets of strings	$L1 - L2$
Complement(L1)	Set complement of L1	$\Sigma^* - L1$

Typical Operations on Languages

Operation name	Properties	Representation
Reverse(L)	Reverse all strings in L	$\{a, aab, abab\}^R = \{a, baa, baba\}$
Concatenation(L1,L2)	The set of strings which has a prefix from L1 and a suffix from L2	L1 = {a, aa,aaa,…} L2 = {b,bb,bbb,…} L1.L2 = {ab,abb,…aab,…}
Kleene Star(L) denoted as L*	The union of concatenations of strings in L1	$(L)^0 \cup (L)^1 \cup (L)^2 \ldots$
Complement(L)	Set complement of L1	$\Sigma^* - L1$

More on Reverse of a Language

Definition: $L^R = \{w^R : w \in L\}$

More Examples:

$$L = \{a^n b^n : n \geq 0\}$$

$$L^R = \{b^n a^n : n \geq 0\}$$

More on Concatenation of Two Languages

Definition: $L_1 L_2 = \{xy : x \in L_1, y \in L_2\}$

Example: $\{a, ab, ba\}\{b, aa\}$

$= \{ab, aaa, abb, abaa, bab, baaa\}$

N-ary Concatenation of a Language

Definition: $L^n = \underbrace{LL\cdots L}_{n}$

$$\{a,b\}^3 = \{a,b\}\{a,b\}\{a,b\} =$$

$$\{aaa, aab, aba, abb, baa, bab, bba, bbb\}$$

Special case: $L^0 = \{\lambda\}$

$$\{a, bba, aaa\}^0 = \{\lambda\}$$

Example

$$L = \{a^n b^n : n \geq 0\}$$

$$L^2 = \{a^n b^n a^m b^m : n, m \geq 0\}$$

$$aabbaaabbb \in L^2$$

Star-Closure (Kleene *)

All strings that can be constructed from L

Definition: $L^* = L^0 \cup L^1 \cup L^2 \ldots$

Example:

$$\{a,bb\}^* = \left\{ \begin{array}{l} \lambda, \\ a, bb, \\ aa, abb, bba, bbbb, \\ aaa, aabb, abba, abbbb, \ldots \end{array} \right\}$$

Positive Closure

Definition: $L^+ = L^1 \cup L^2 \cup \cdots$

$$\{a, bb\}^+ = \left\{ \begin{array}{l} a, bb, \\ aa, abb, bba, bbbb, \\ aaa, aabb, abba, abbbb, \ldots \end{array} \right\}$$

The * Operation on alphabets

Σ^*: the set of all possible strings from alphabet Σ

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

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

The + Operation on alphabets

Σ^+ : the set of all possible strings from alphabet Σ except λ

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

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

$$\Sigma^+ = \Sigma^* - \lambda$$

$$\Sigma^+ = \{a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

Two special languages

Empty language

$\{\ \}$

Language with empty string

$\{\lambda\}$

Size of a language (number of elements):

$|\{\}| = 0$

$|\{\lambda\}| = 1$

$|\{a, aa, ab\}| = 3$

$|\{\lambda, aa, bb, abba, baba\}| = 5$

Connection between Formal Languages and Computation

Languages are used to describe computational problems:

For example, consider the PRIMES problem is "given a natural number X decide whether X is a PRIME?"

- Very famous problem that was recently shown to be in the complexity class P.

Connection between Formal Languages and Computation

All Computational problems can be described as Language or Set membership problems, i.e., "given a string X, and a language S, does there exist an algorithm to decide whether X belong to S?"

The PRIMES problem can be equivalently written as a set membership problem over the language of strings over digits:

- Alphabet: {0,1,2,…,9}
- Example Strings: 0, 100,..
- Language of PRIMES: {2,3,5,7,…}

Question: How can the compilation problem be described using the paradigm of language membership?

Languages, Problems and Their Solutions

Problem S: Computational problems can be described as Language or Set membership problems, i.e., "given a string X, and a language S, does there exist an algorithm/function/method to decide whether X belongs to S?"

(Side Note: We can always reduce optimization problems to membership/decision problems.)

Solution to S: The algorithm C that takes as input any X and correctly decides whether X belongs to S is said to be a solution to the problem S.

However, we need to be a bit more precise about what we mean by an algorithm C, and the possible outputs C can produce for a problem S.

Languages and Turing Machines

Enter Alan Turing:

- Proposes the Turing Machine (TM)
- Establishes the existence of an Universal Turing Machine
- Any computable function (or method or algorithm) can be implemented as a "program" on TM

Solution to S in terms of Turing Machines: We say computational problem S has a solution if there exists a Turing Machine P that correctly decides for any string X whether X belongs to S.

We need one some more precision in our description of P:

- P can correctly say X in S and HALT
- P can correctly say X is not in S and HALT
- P can loop forever
- (P can also have "bugs" and produce incorrect results. We will ignore this case for now.)

Decidability of Languages

Definition of Turing-acceptable languages: We say a language L is Turing-Acceptable if there exists a Turing Machine P that given any w determines if w belongs to L. We say P accepts L.

More precisely, for any string :

$w \in L$ ⇨ P halts in an accept state

$w \notin L$ ⇨ P halts in a non-accept state or does not terminate

Decidability of Languages

Definition of decidable languages: We say a language L is decidable if there exists a Turing Machine P that given any w determines if w belongs to L or not.

More precisely, for any string :

$w \in L$ $\Rightarrow$ P halts in an accept state

$w \notin L$ $\Rightarrow$ P halts in a non-accept state

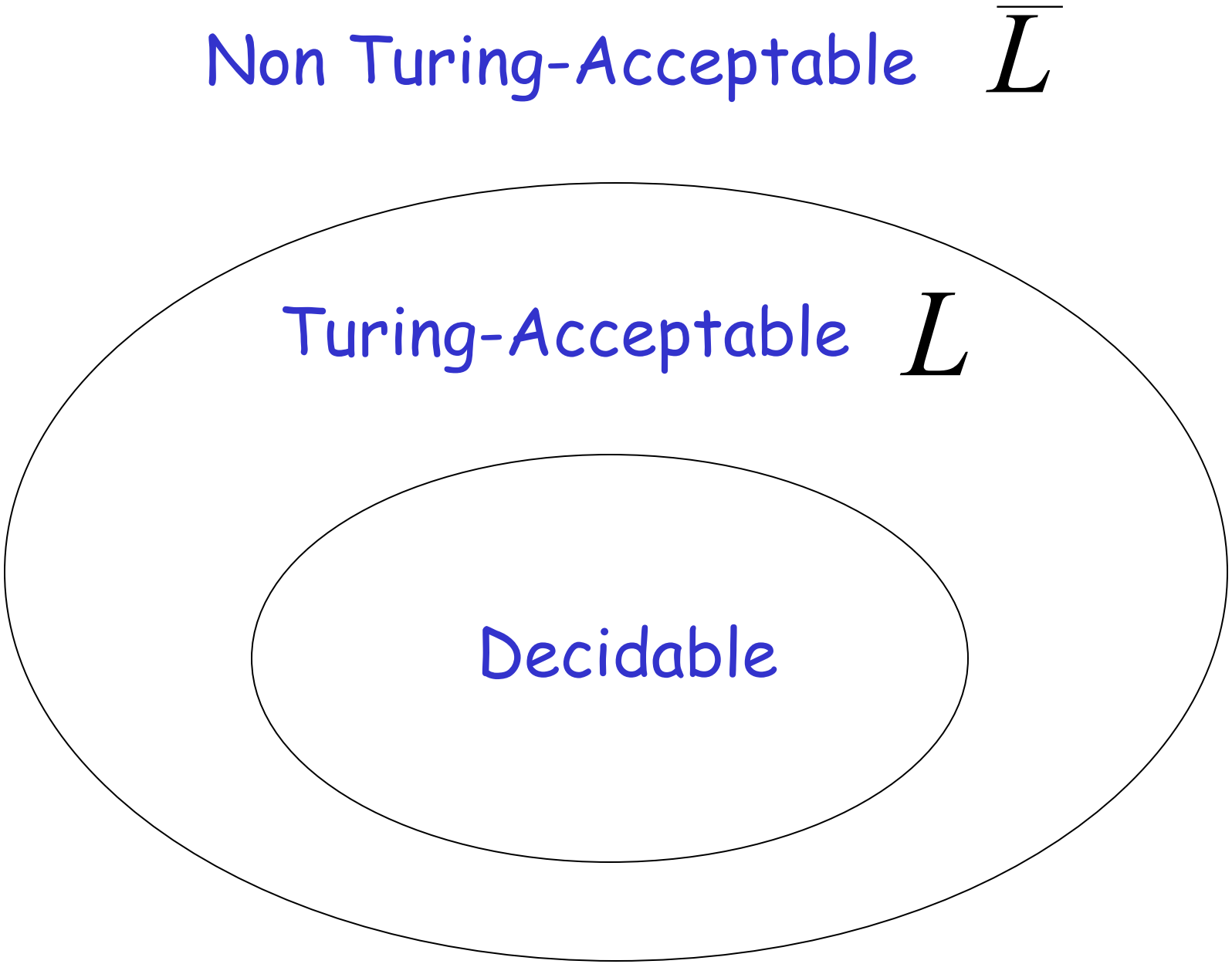
Side Note: Languages that are not decidable are called undecidable.

Connect between Turing-acceptable and decidable languages

Theorem: Every decidable language is Turing-acceptable.

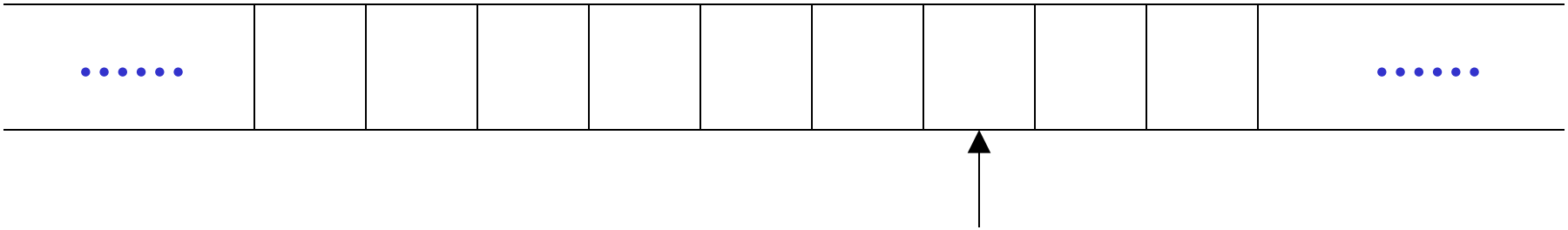
However, not every Turing-acceptable language is decidable. For example, the halting problem.

(Side note: Turing-acceptable languages are also called recursively-enumerable.)


Non Turing-Acceptable
$\overline{L}$
Turing-Acceptable
L
Decidable

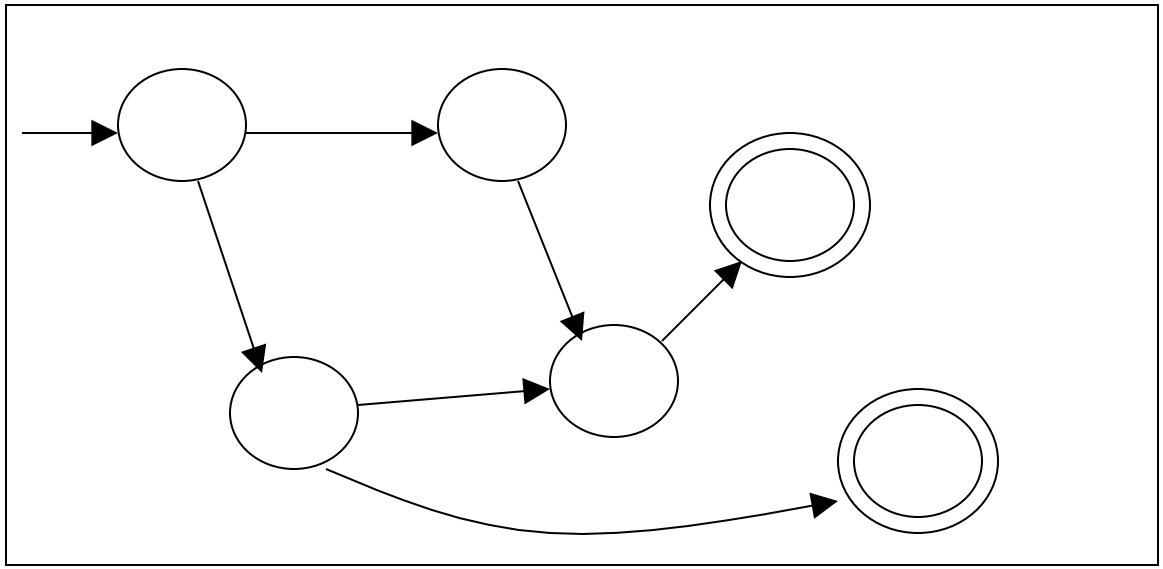
A Turing Machine

Tape



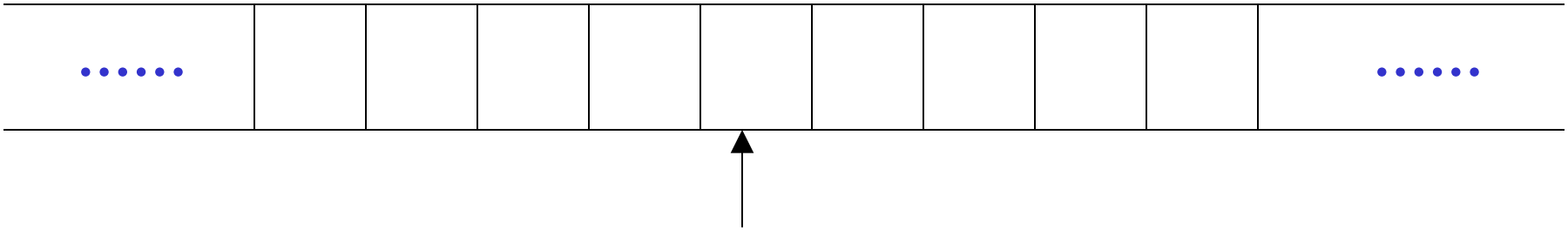
Read-Write head

Control Unit



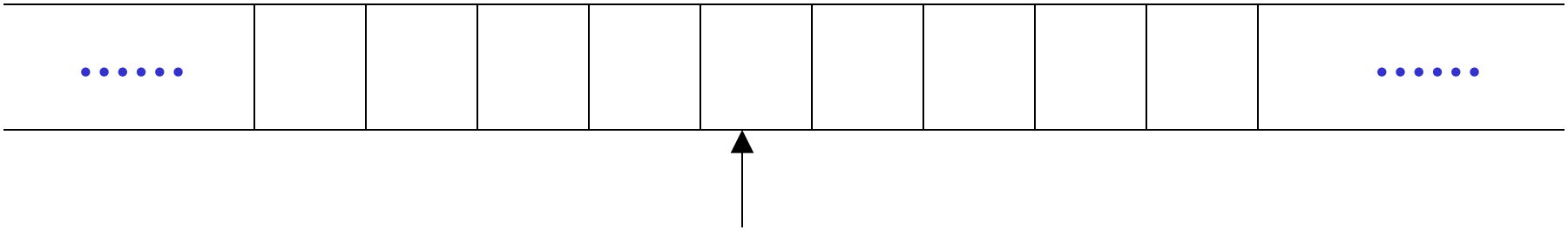
The Tape

No boundaries -- infinite length



Read-Write head

The head moves Left or Right



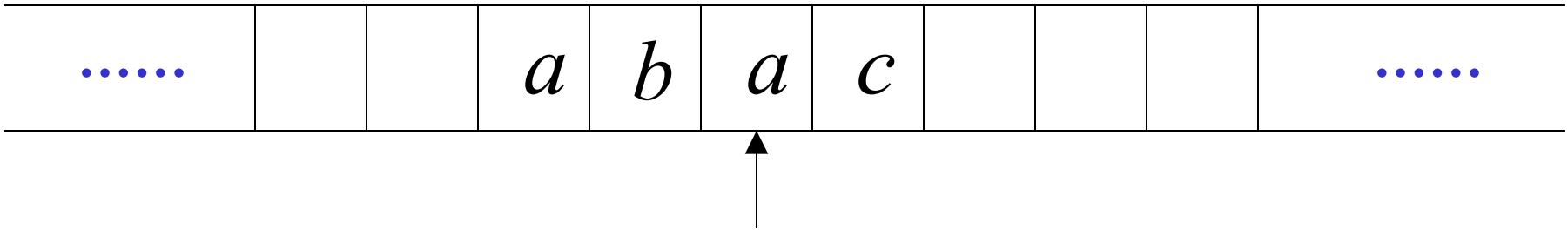
Read-Write head

The head at each transition (time step):

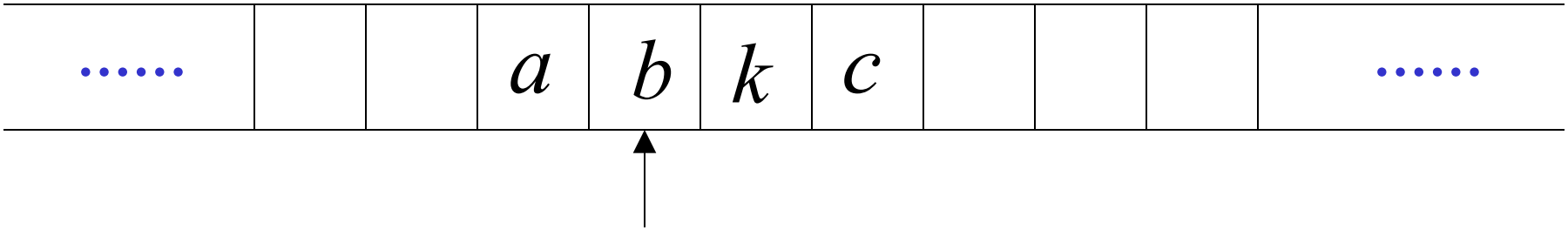
1. Reads a symbol
2. Writes a symbol
3. Moves Left or Right

Example:

Time 0

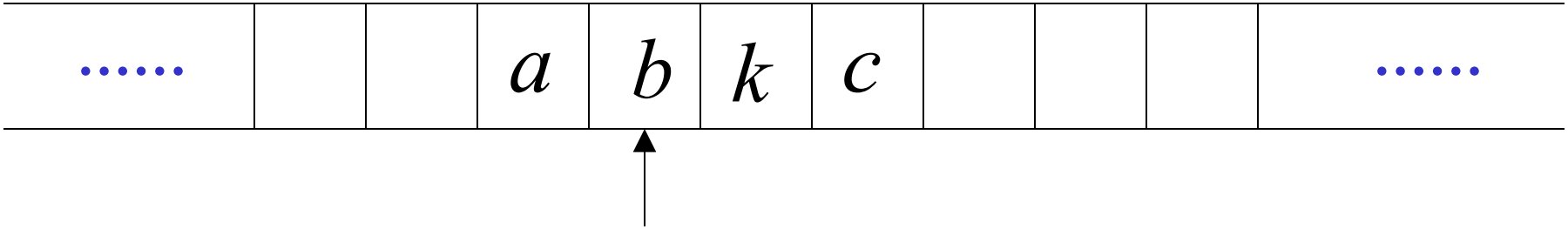


Time 1

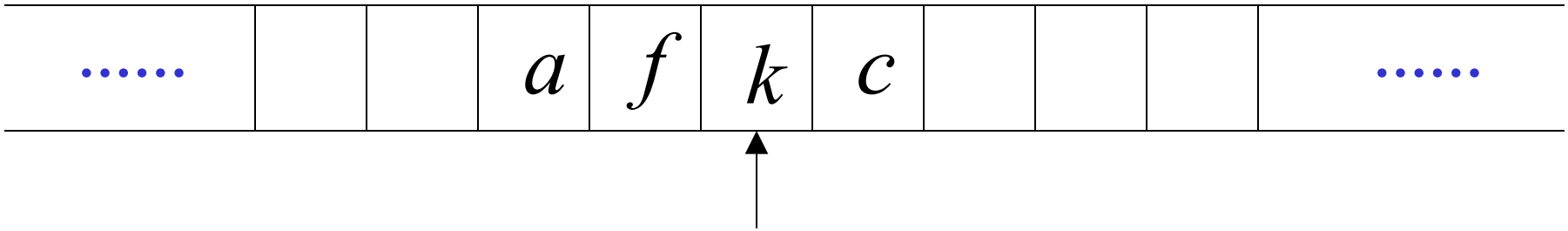


1. Reads a
2. Writes k
3. Moves Left

Time 1

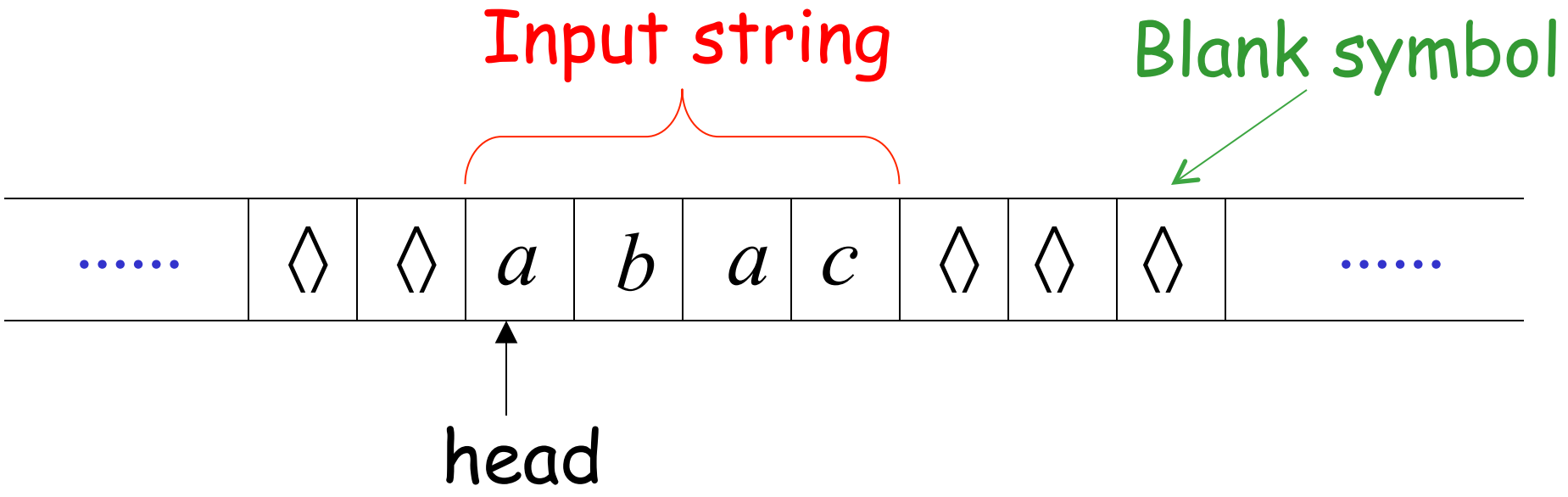


Time 2



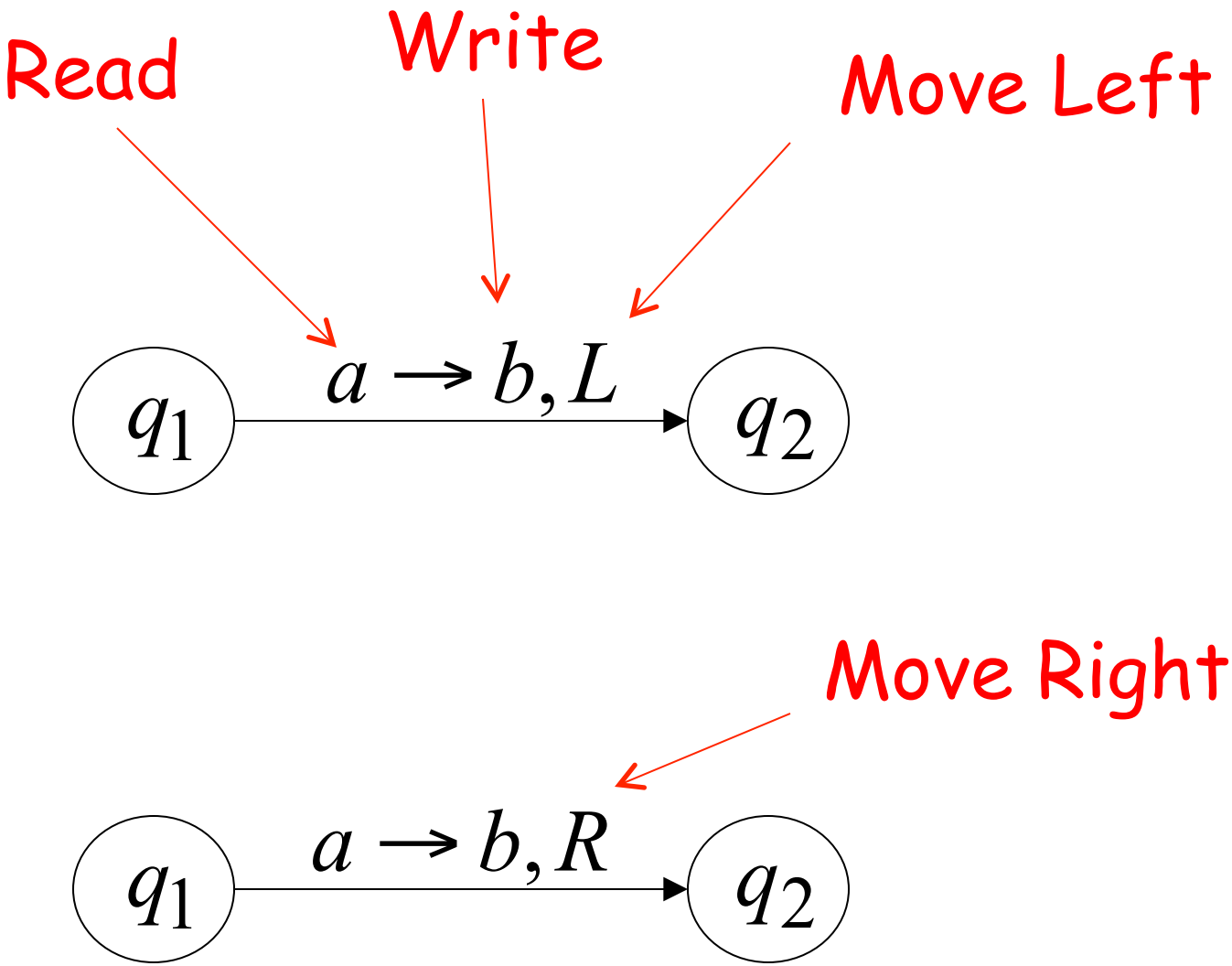
1. Reads b
2. Writes f
3. Moves Right

The Input String



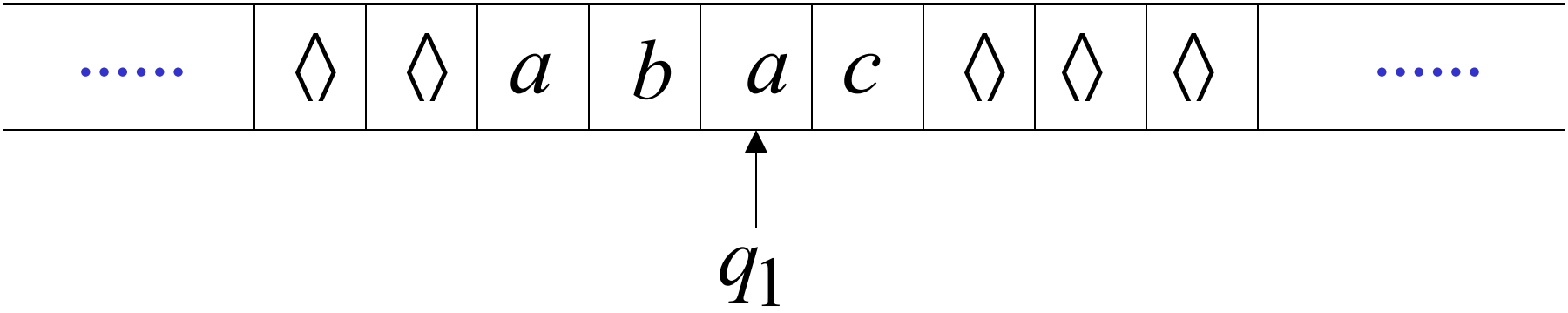
Head starts at the leftmost position of the input string

States & Transitions

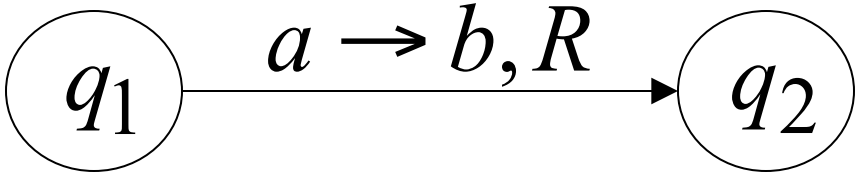


Example:

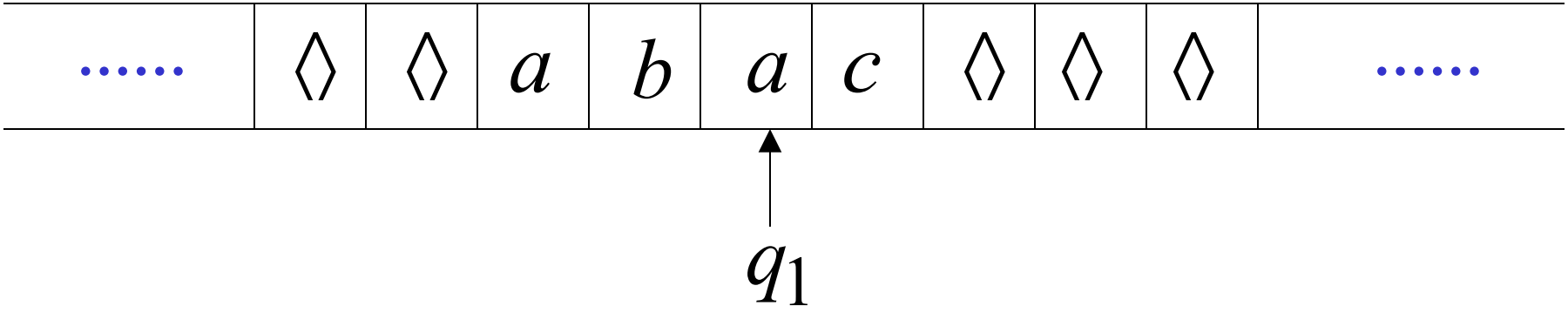
Time 1



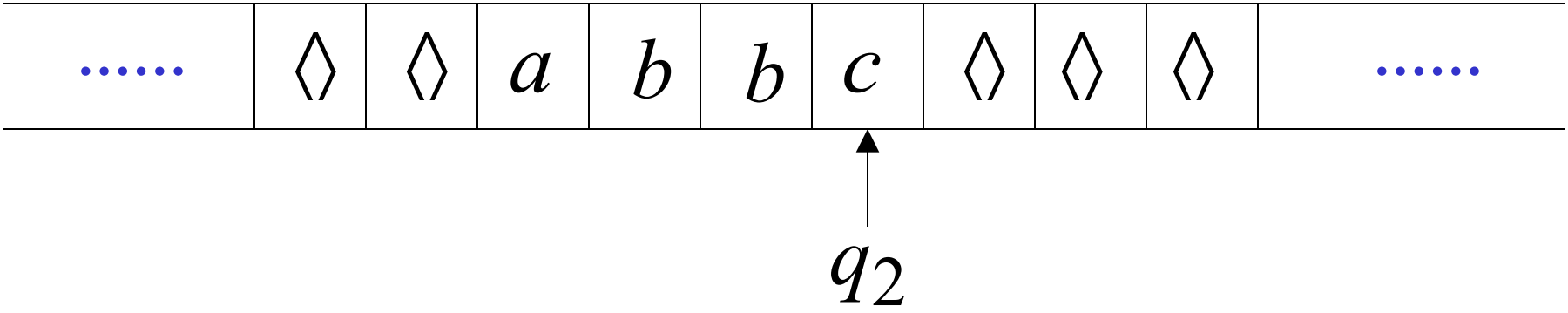
current state

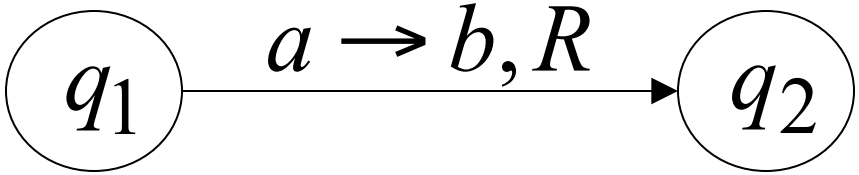


Time 1



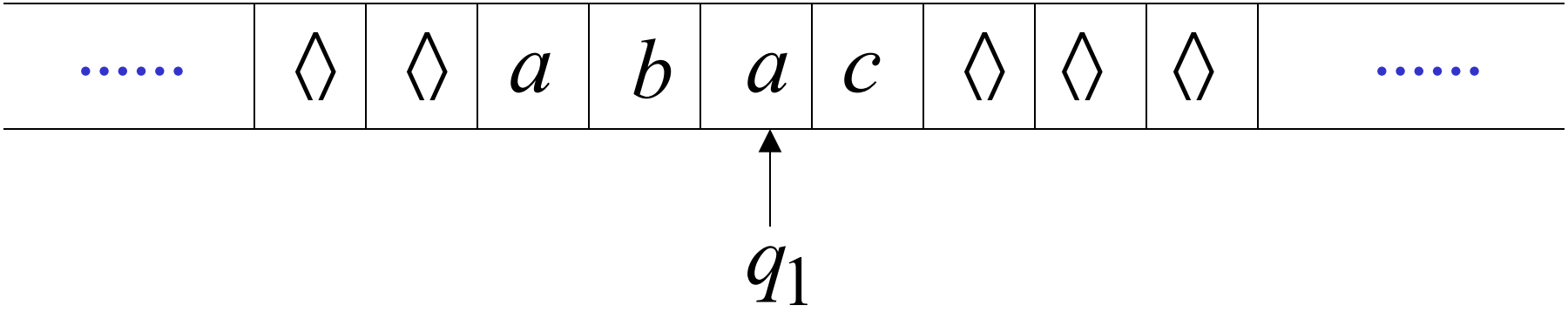
Time 2



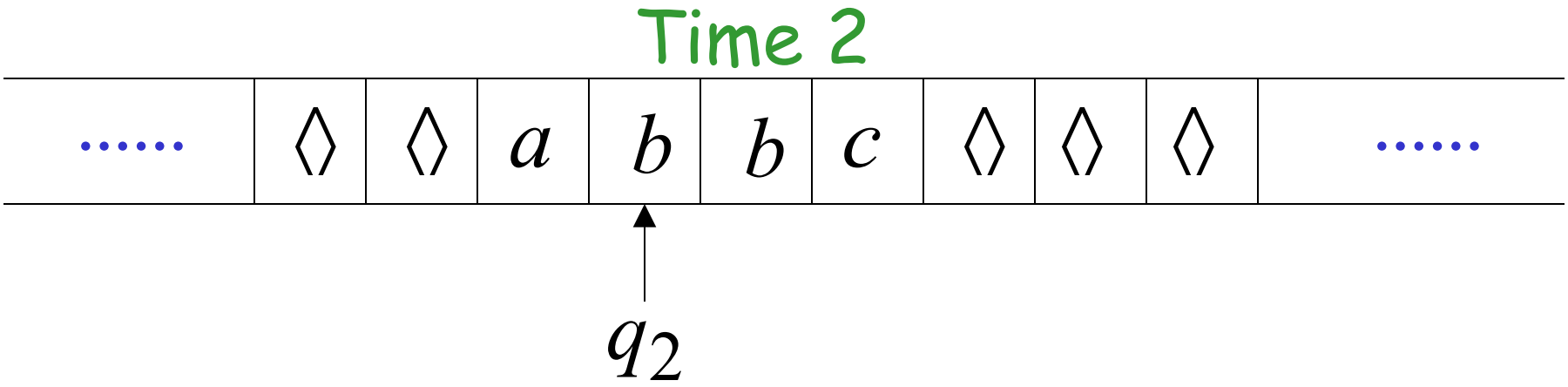


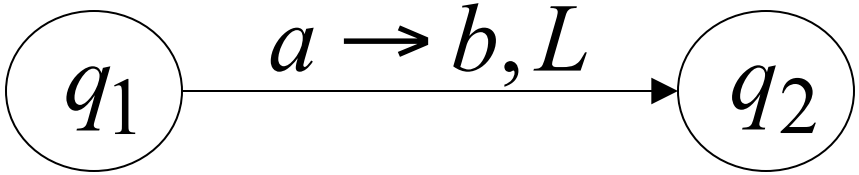
Example:

Time 1



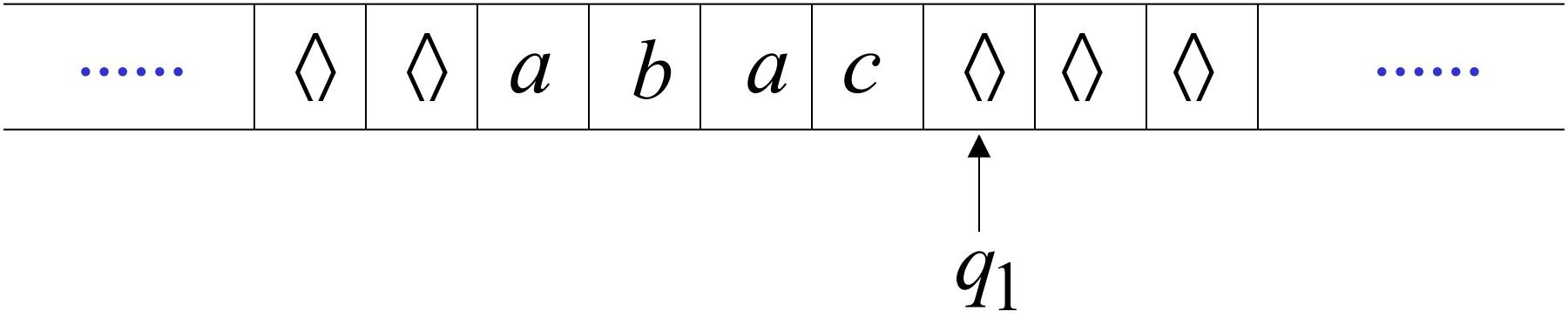
Time 2



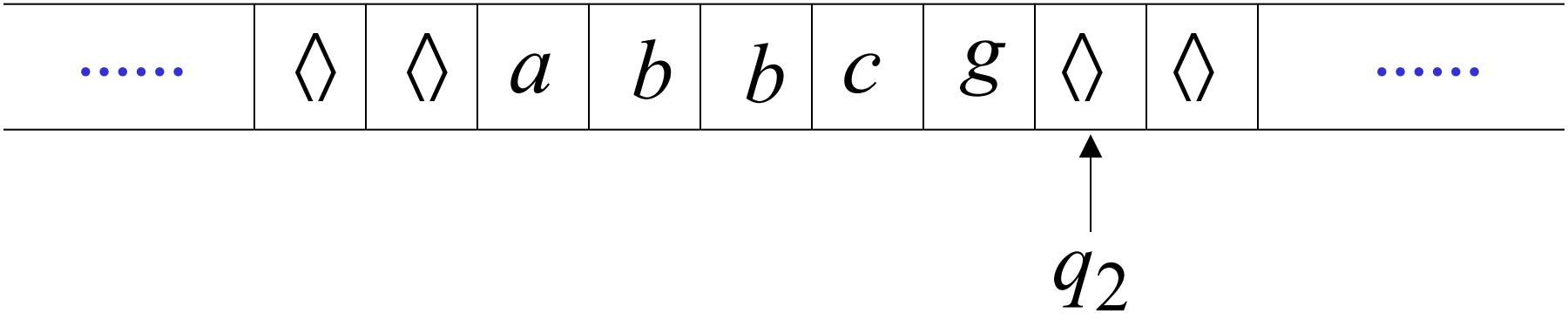


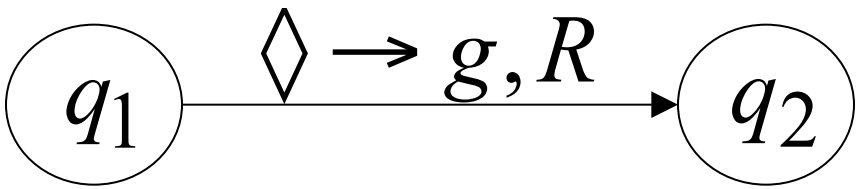
Example:

Time 1



Time 2

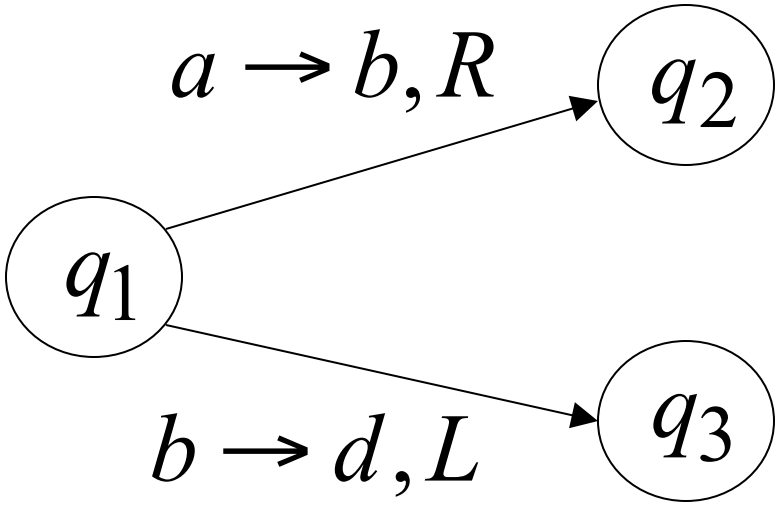




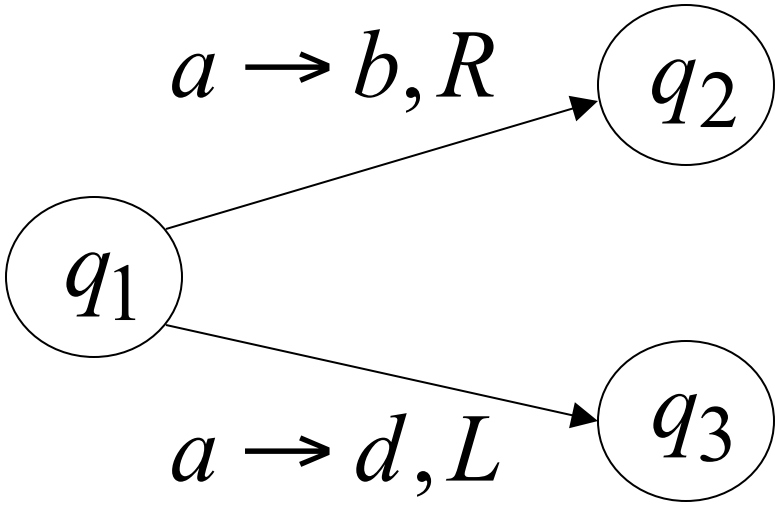
Determinism

Turing Machines as originally defined are deterministic

Allowed



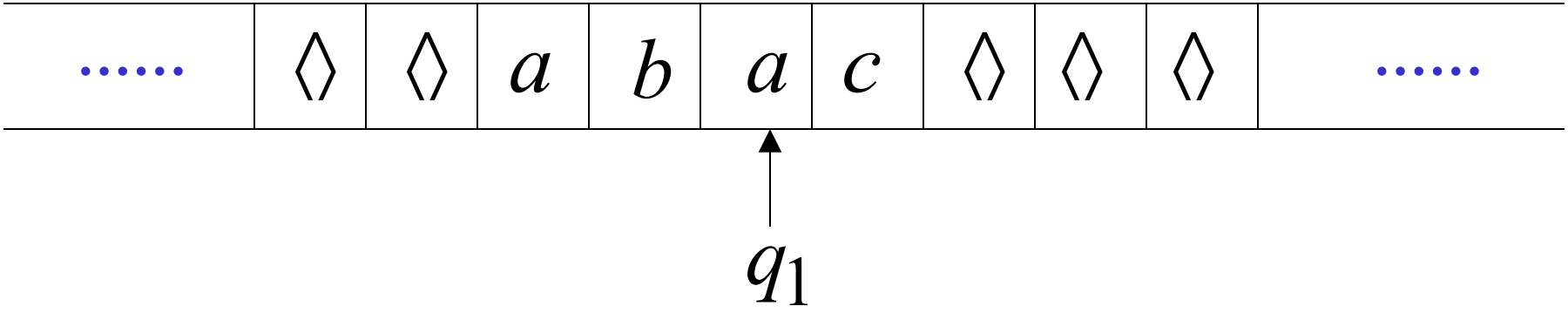
Not Allowed

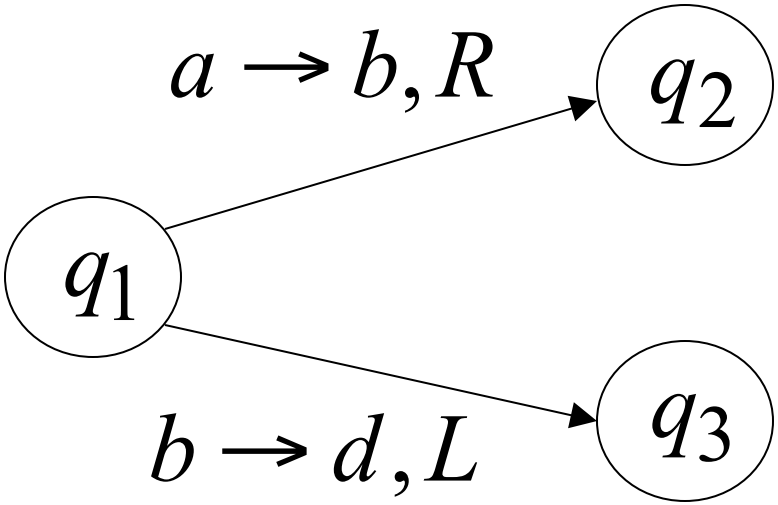


No lambda transitions allowed

Partial Transition Function

Example:





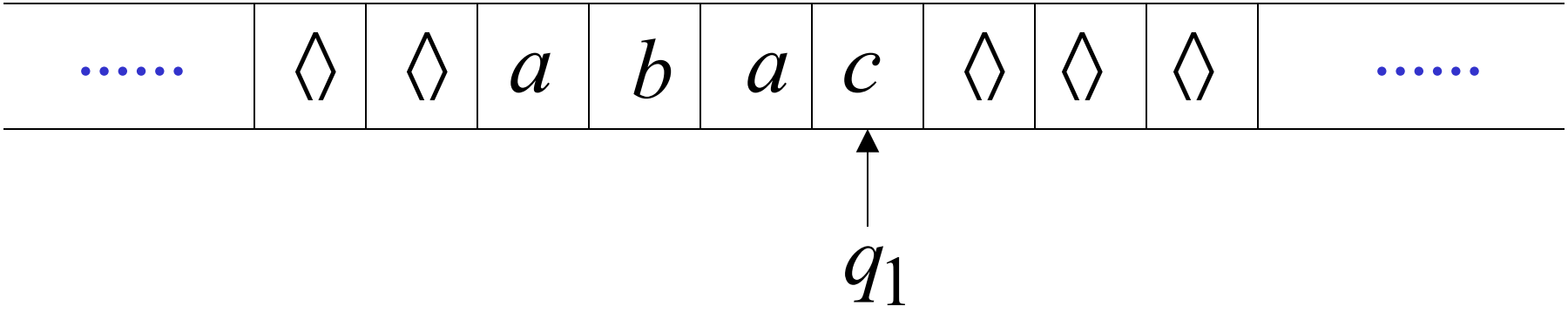
Allowed:

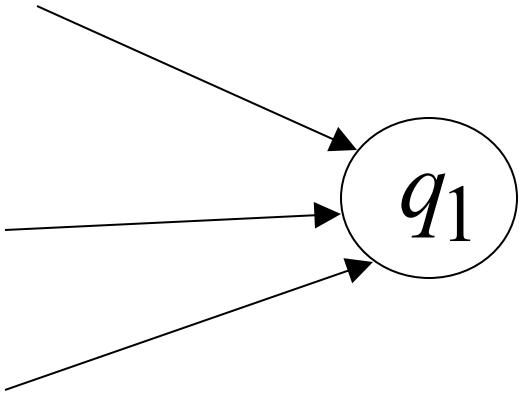
No transition for input symbol c

Halting

The machine ***halts*** in a state if there is no transition to follow or out of the state

Halting Example 1:

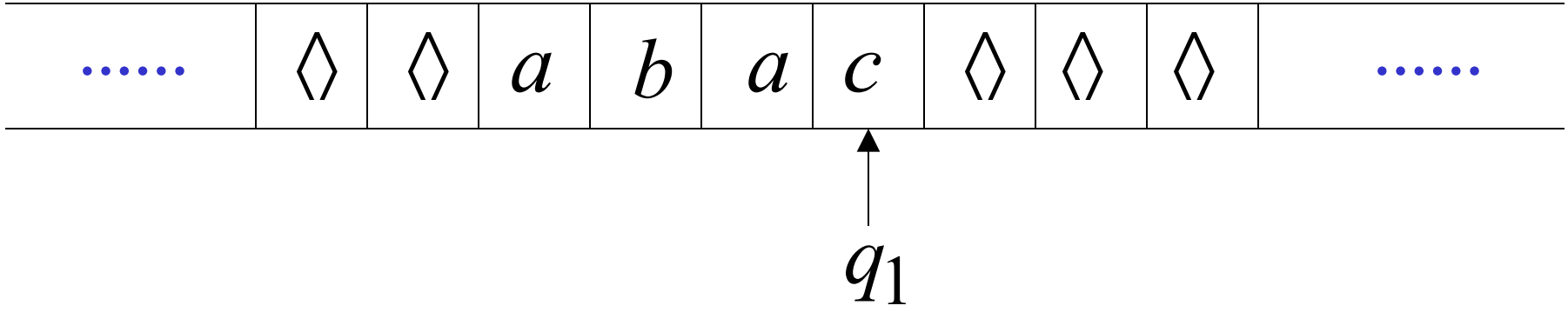


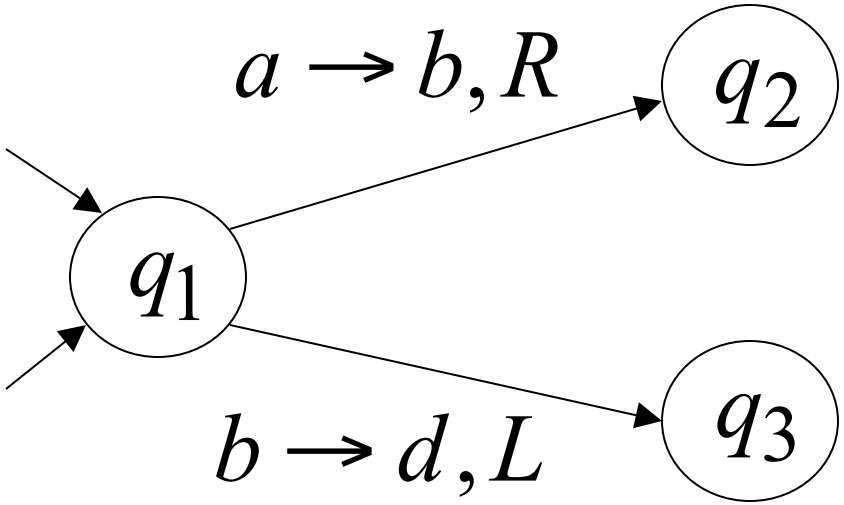


No transition from q_1

HALT!!!

Halting Example 2:

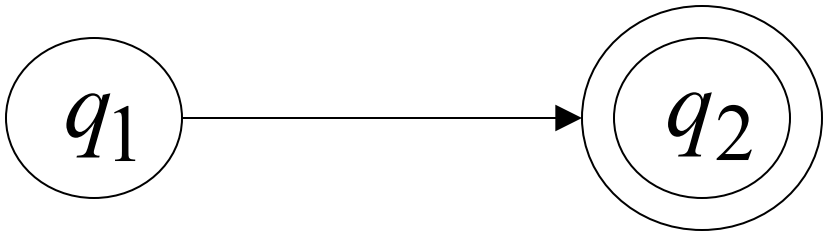




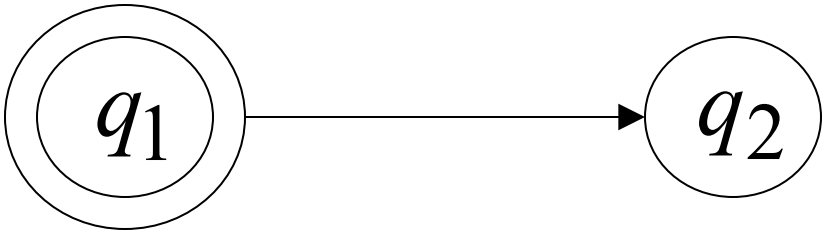
No possible transition from q_1 and symbol c

HALT!!!

Accepting States



Allowed

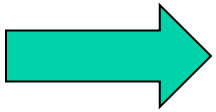


Not Allowed

- The machine accepts and halts

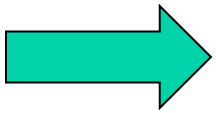
Acceptance

Accept Input string



If machine halts in an accept state

Reject Input string



If machine halts in a non-accept state

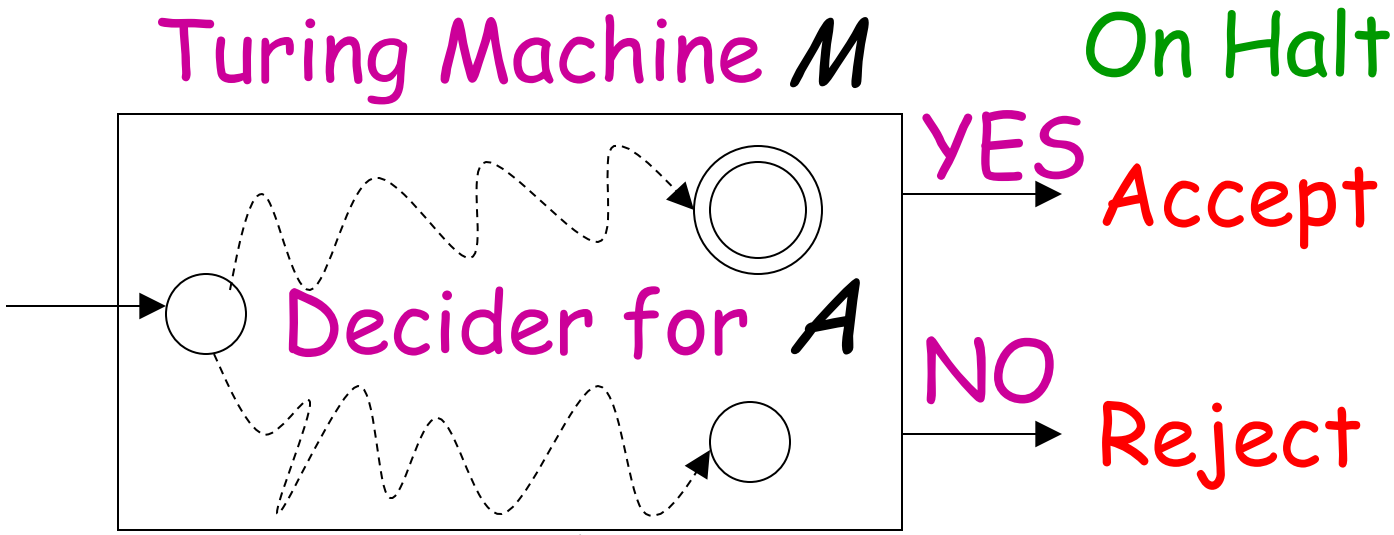
or

If machine enters an *infinite loop*

Decidable Languages

Recall that:

A language A is decidable,
if there is a Turing machine M (decider)
that accepts the language A and
halts on every input string



Recall: A computational problem is decidable if the corresponding language is decidable

We also say that the problem is solvable

Halting Problem

Input:
- Turing Machine M
- String w

Question: Does M halt while processing input string w ?

Corresponding language:

$$HALT_{TM} = \{\langle M, w\rangle : M \text{ is a Turing machine that halts on input string } w\}$$

The Diagonalization proof

Theorem: $HALT_{TM}$ is undecidable

(The halting problem is unsolvable)

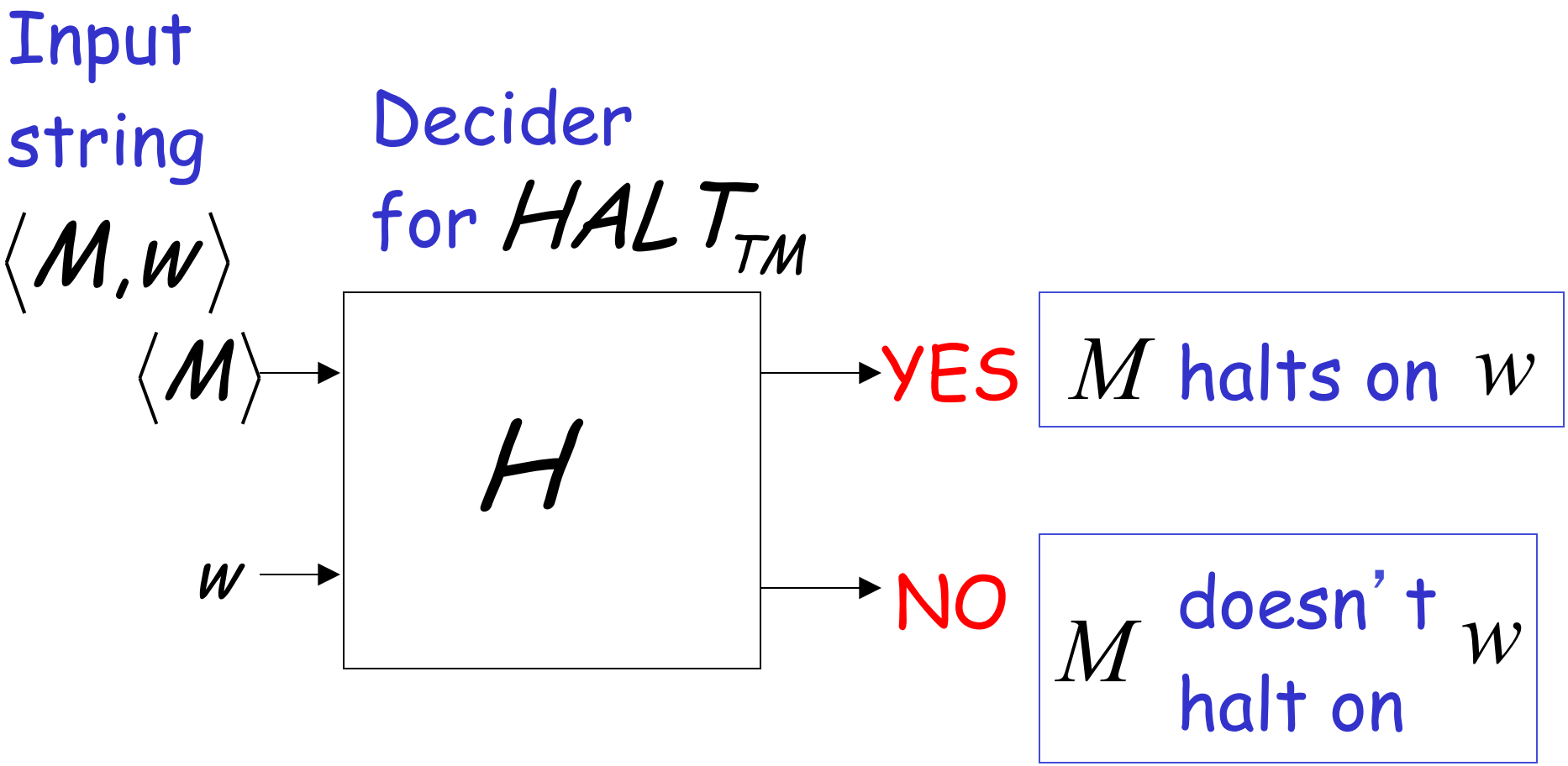
Proof:

Basic idea:

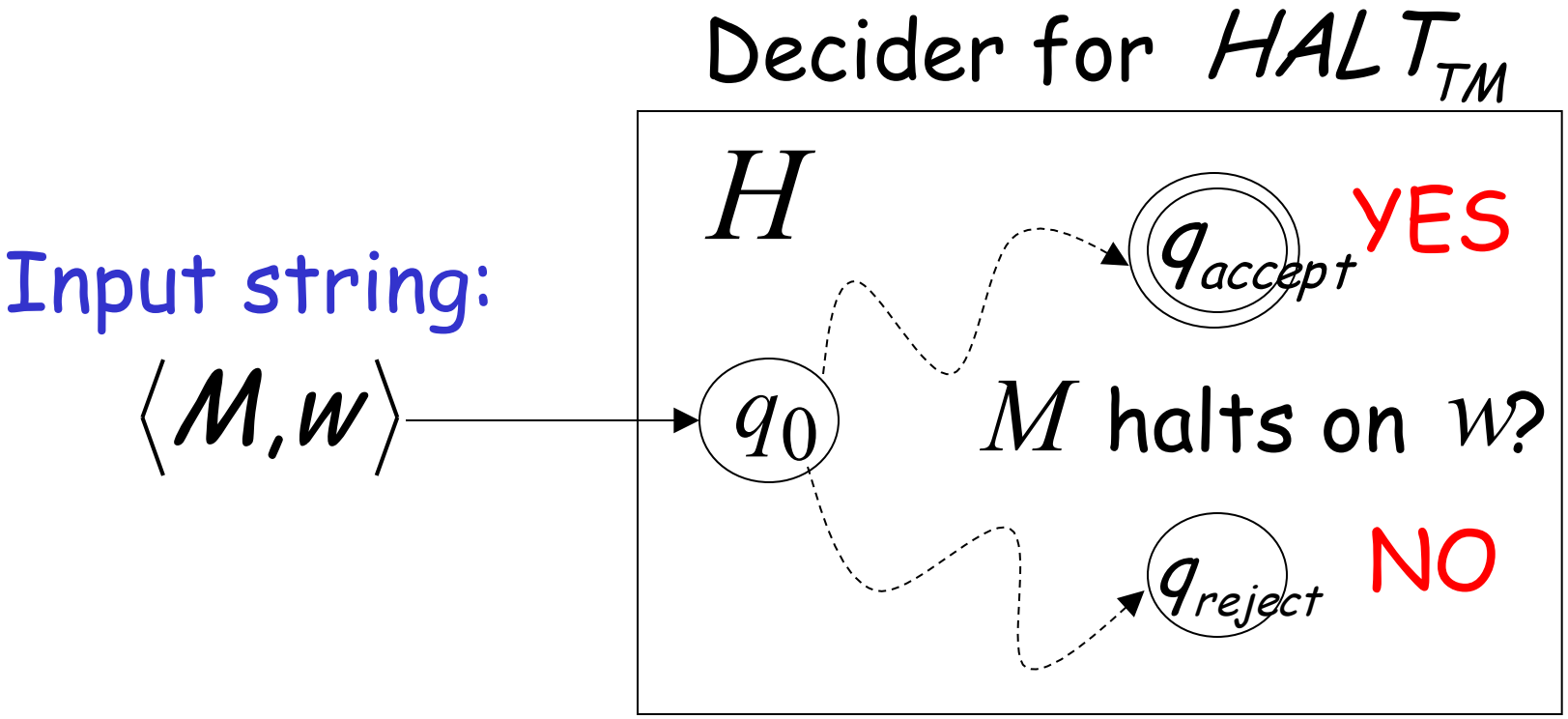
Assume for contradiction that the halting problem is decidable;

We will obtain a contradiction using a diagonalization technique

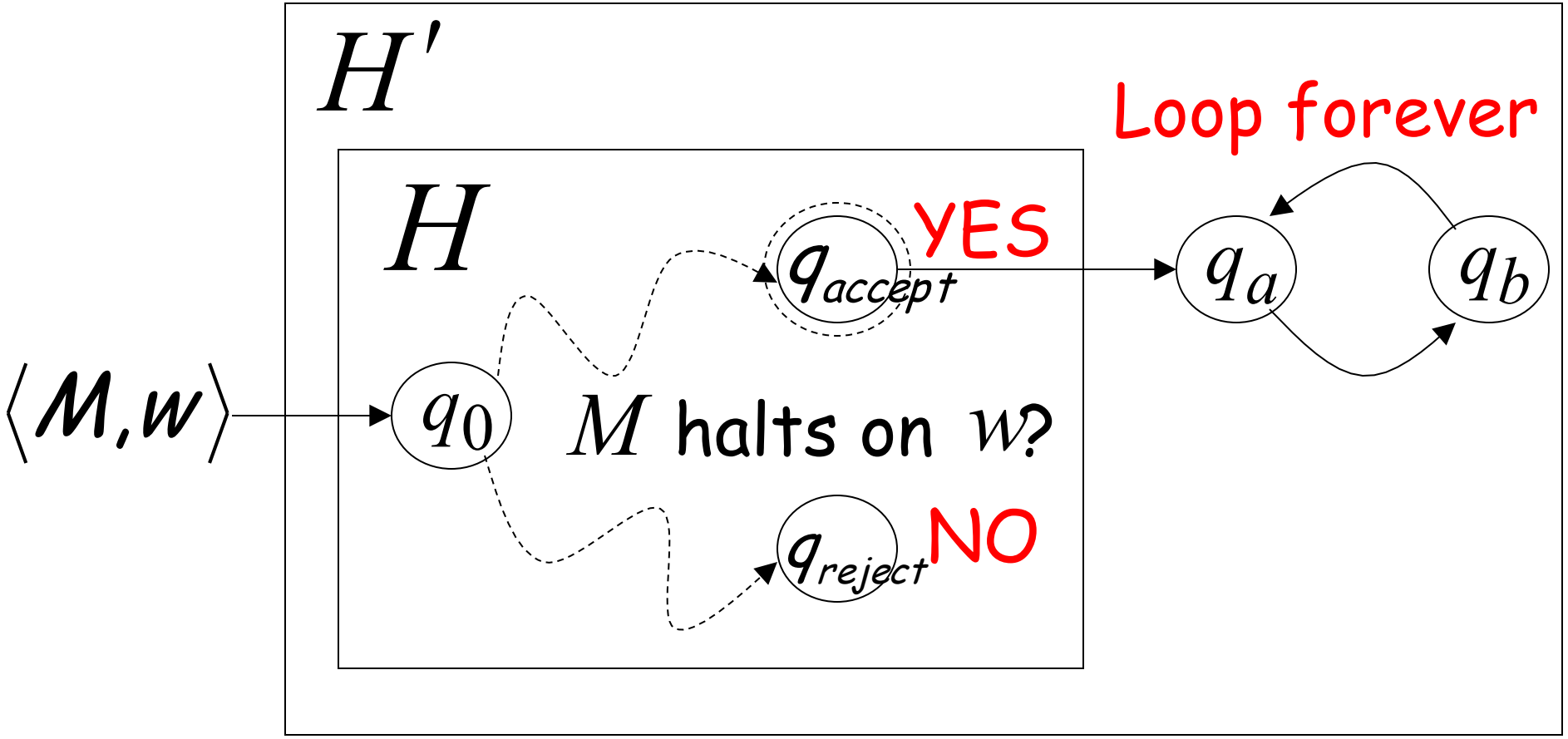
Suppose that $HALT_{TM}$ is decidable



Looking inside H



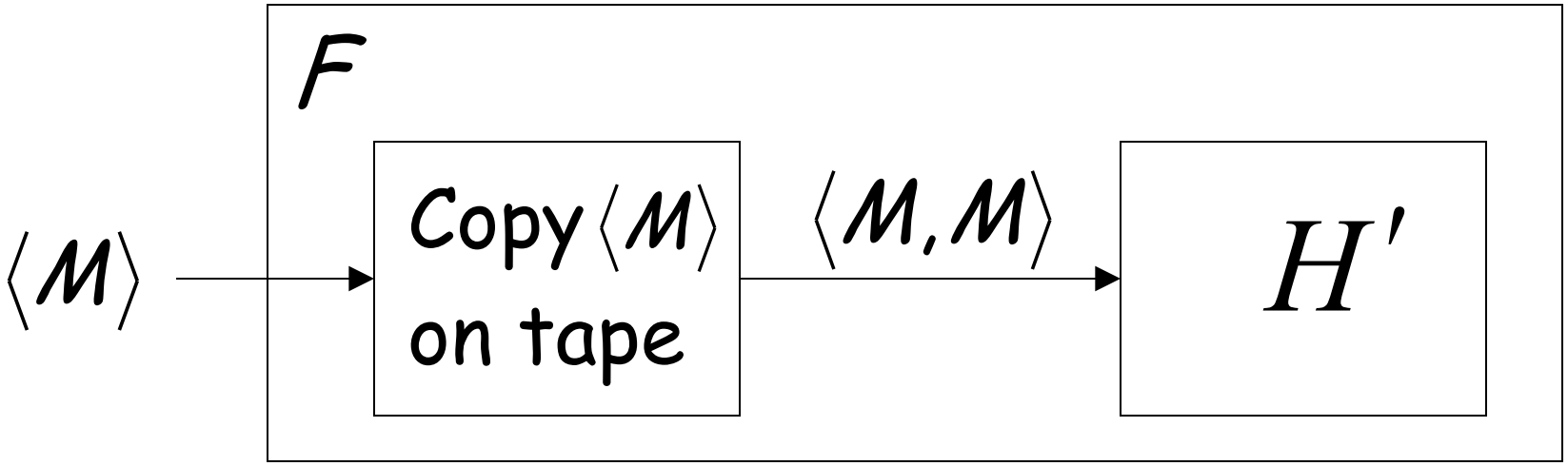
Construct machine H' :



If M halts on input w **Then** Loop Forever

Else Halt

Construct machine F :

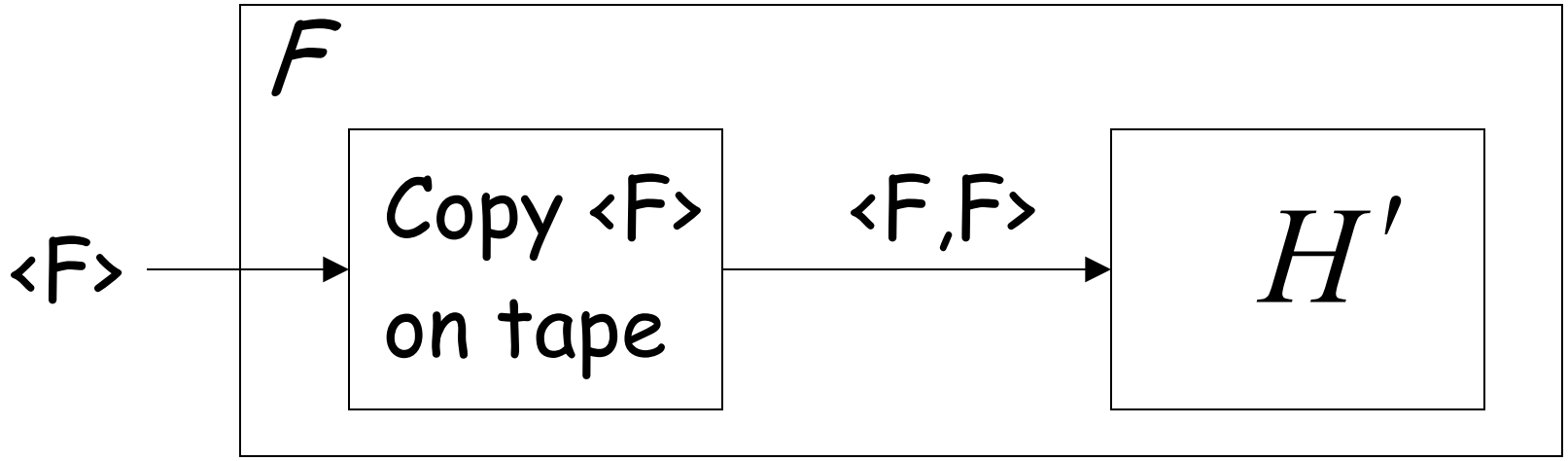


If F halts on input $\langle M \rangle$

Then loop forever

Else halt

Run F with input itself



If (F halts on input <F>)

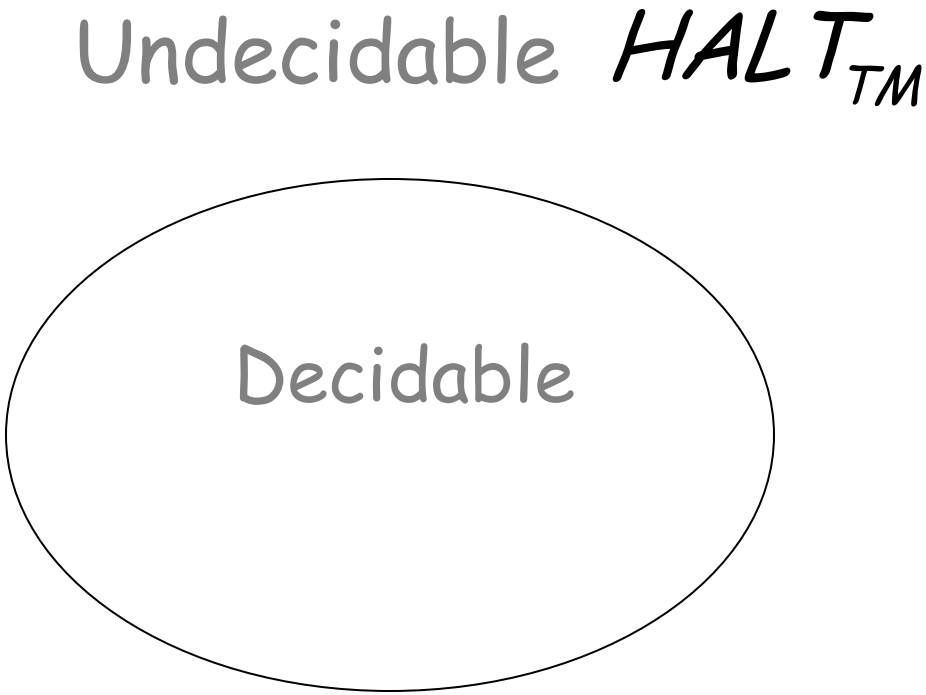
Then F loops forever on input <F>

Else (F does not halt <F>) F halts on input <F>

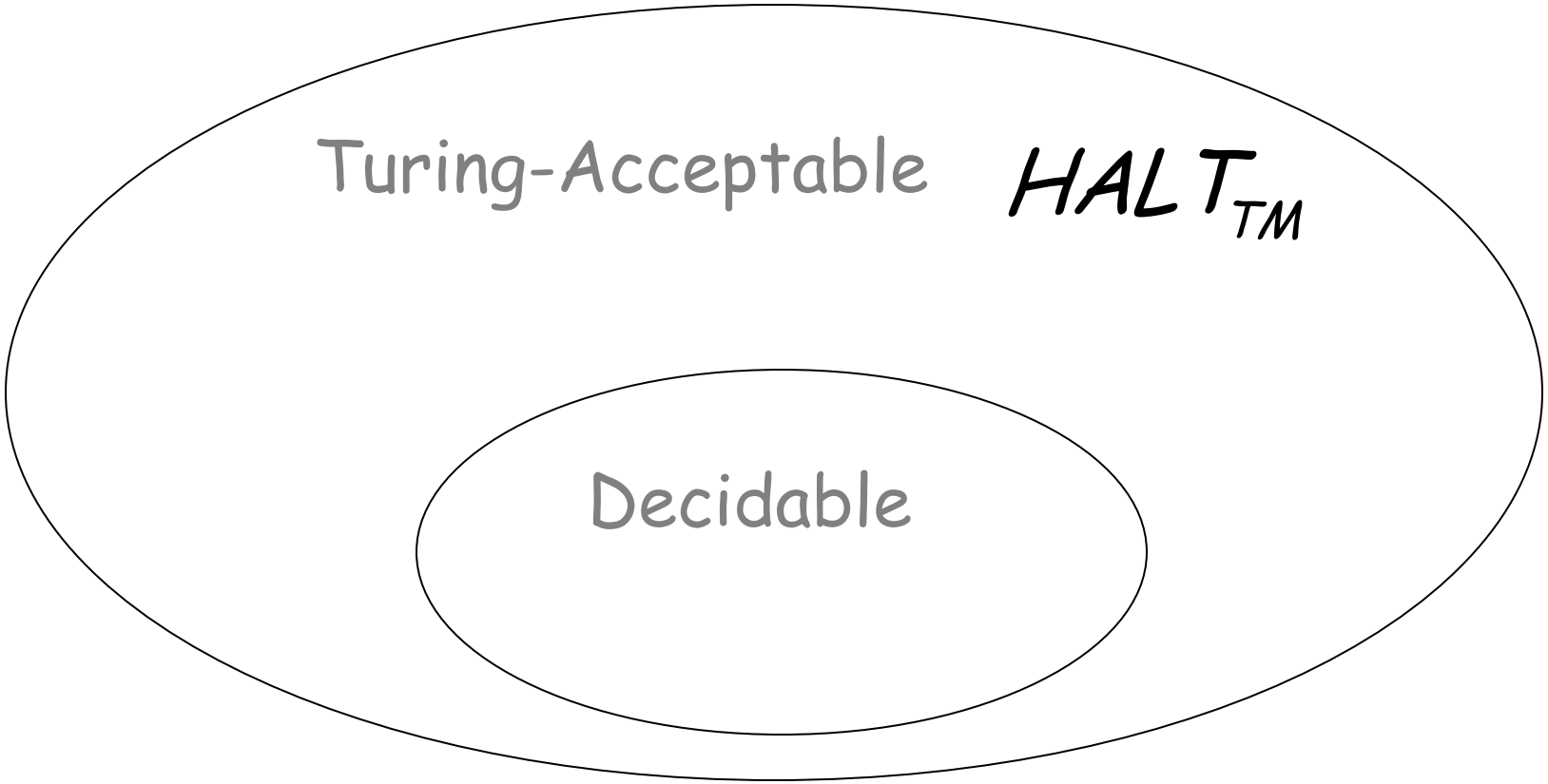
CONTRADICTION!!!

END OF PROOF

We have shown:

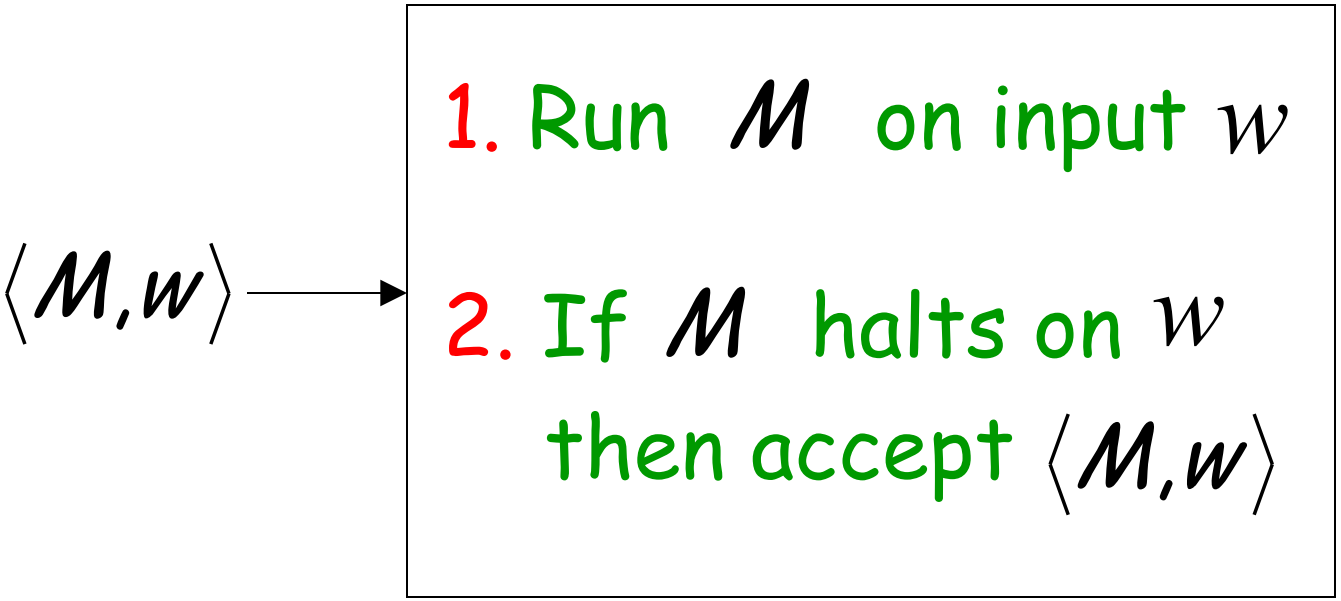


We can actually show:



$HALT_{TM}$ is Turing-Acceptable

Turing machine that accepts $HALT_{TM}$:



Static Program Analysis is Undecidable

- We can show that static analysis is Turing-acceptable. (Informal statement of Rice's Theorem)

- Informal Proof by reduction:

  Let say that static analysis is decidable. It immediately follows that we can easily analyze any program and determines if it halts on an input. If so we have solved the halting problem.

  CONTRADICTION!!