The Turing Machine (Algorithms Part III)

ver. 11 z drobnymi modyfikacjami!

Wojciech Myszka

2023-11-28 07:32:04 +0100

ocław University
of Science and Technology

Introduction

This lecture is rather difficult.
I would like to present you algorithmic devices of the simplest imaginable kind, strikingly primitive in contrast with today's computers and programming languages. Nevertheless, they are powerful enough to execute even the most complex algorithms.

Introduction

This lecture is rather difficult.
I would like to present you algorithmic devices of the simplest imaginable kind, strikingly primitive in contrast with today's computers and programming languages. Nevertheless, they are powerful enough to execute even the most complex algorithms.

- Computers are becoming faster and faster and more sophisticated by the year.
- It is very interesting to discover objects that are as simple as possible yet as powerful as anything of their kind.

The idea of data

- Any data item used by an algorithm, whether as an input, output or intermediate value, can be thought of as a string of symbols.
- Integer number is but a string of digits.
- Fractional number can be defined as two strings of digits separated by slash.
- Word in English (or any other language) is string of letters; an entire text is a string of symbols consisting of letter, spaces, punctuation marks, etc.
- And so on…
- The number of different symbols is finite

The idea of data

- Any data item used by an algorithm, whether as an input, output or intermediate value, can be thought of as a string of symbols.
- Integer number is but a string of digits.
- Fractional number can be defined as two strings of digits separated by slash.
- Word in English (or any other language) is string of letters; an entire text is a string of symbols consisting of letter, spaces, punctuation marks, etc.
- And so on...
- The number of different symbols is finite

Consequently, we can write any data item of interest along a one-dimensional tape, perhaps a long one, which consists of a sequence of squares, each containing a single symbol that is a member of some finite alphabet.

Simplifying data

- A vector, for example can be depicted as a sequence of the linearised versions of each of the items, separated by a special symbol, such as "*".
- A two-dimensional array can be spread out row by row along the tape, using "*" to separate items within each row and, say, "**" to separate rows

Example

Simplifying data

- A vector, for example can be depicted as a sequence of the linearised versions of each of the items, separated by a special symbol, such as "*".
- A two-dimensional array can be spread out row by row along the tape, using "*" to separate items within each row and, say, "**" to separate rows

Example

11	12	13
21	22	23
31	32	33
41	42	43

Simplifying data

- A vector, for example can be depicted as a sequence of the linearised versions of each of the items, separated by a special symbol, such as "*".
- A two-dimensional array can be spread out row by row along the tape, using "*" to separate items within each row and, say, "**" to separate rows

Example

11	12	13
21	22	23
31	32	33
41	42	43

Can be written as: 11* 12* 13 ** 21* 22* 23 ** 31* 32* 33 ** 41* 42* 43 **

Simplifying data

- A vector, for example can be depicted as a sequence of the linearised versions of each of the items, separated by a special symbol, such as "*".
- A two-dimensional array can be spread out row by row along the tape, using "*" to separate items within each row and, say, "**" to separate rows

Example

11	12	13
21	22	23
31	32	33
41	42	43

Can be written as: 11* 12* 13 ** 21* 22* 23 ** 31* 32* 33 ** 41* 42* 43 **
or as: 11; 21; 31; 41 * 12; 22; 32; 42 * 13; 23; 33; 43 *

Simplifying data

- A vector, for example can be depicted as a sequence of the linearised versions of each of the items, separated by a special symbol, such as "*".
- A two-dimensional array can be spread out row by row along the tape, using "*" to separate items within each row and, say, "**" to separate rows

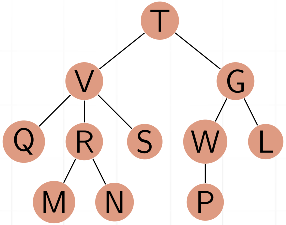
Example

11	12	13
21	22	23
31	32	33
41	42	43

Can be written as: 11* 12* 13 ** 21* 22* 23 ** 31* 32* 33 ** 41* 42* 43 **
or as: 11; 21; 31; 41 * 12; 22; 32; 42 * 13; 23; 33; 43 *
or as: {{11, 21, 31, 41}, {12, 22, 32, 42}, {13, 23, 33, 43}}, i.e., as a list.

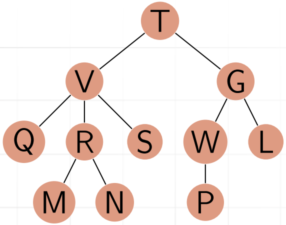
Simplifying data

- What about a stack (LIFO) and/or a queue(FIFO)?
- Database: it is kind of big table...
- And a tree...?



Simplifying data

Tree



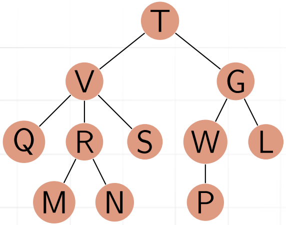
If we attempt to naively list the tree's items level by level, the precise structure of the tree may be lost, since the number of items on a given level is not fixed:

T ** V; G ** Q; R; S; W; L ** M; N; P**

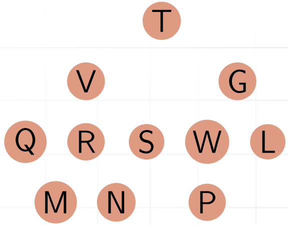
(stars mark the "end of each level") But one thing that can be recovered is...

Simplifying data

Tree

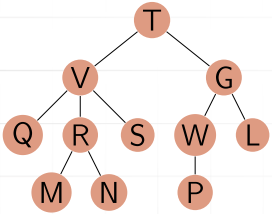


If we attempt to naively list the tree's items level by level, the precise structure of the tree may be lost, since the number of items on a given level is not fixed:
T ** V; G ** Q; R; S; W; L ** M; N; P**
(stars mark the "end of each level") But one thing that can be recovered is...



Simplifying data

Tree



One way of avoiding the problem is to adopt a variant of the "nested lists." Alternatively, we can refine the method by marking off clusters of immediate offspring, level by level, always starting at the left. Here is the resulting linearisation for the tree of
(T) (V; G) (Q; R; S) (W; L) () (M; N) () (P) ()
(The parentheses are considered as special symbols, like "*" and "**".)
Notation is very compact and somehow difficult to read (decode), but allows for linearisation.

Simplifying data — the thesis

The thesis

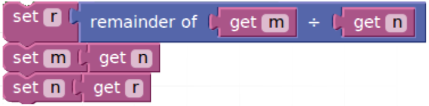
We assume that any data structure can be stored in a linear form, for example, respectively composed of a long tape which consists of a sequence of squares, each containing a single symbol that is a member of some finite alphabet.

Simplifying Control

- What the computer program is?

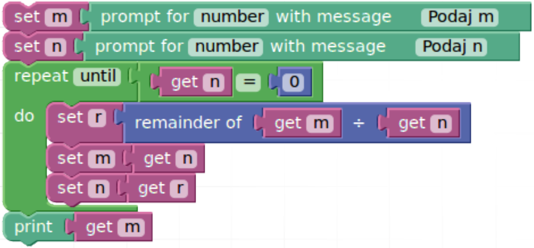
Simplifying Control

- What the computer program is?



Simplifying Control

- What the computer program is?



Simplifying Control

- What the computer program is?
  It is a kind of "control structure" defining the order and kind of operations performed on (input) data.

Simplifying Control

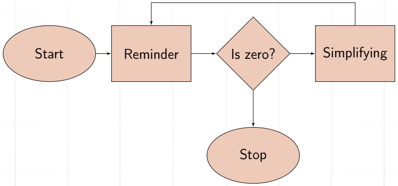
- What the computer program is?
  It is a kind of "control structure" defining the order and kind of operations performed on (input) data.
- Each algorithm is finite.

Simplifying Control

- What the computer program is?
  It is a kind of "control structure" defining the order and kind of operations performed on (input) data.
- Each algorithm is finite.
- Transition between instructions (states) depends on a current state and on the value(s) of certain data items.

Simplifying Control

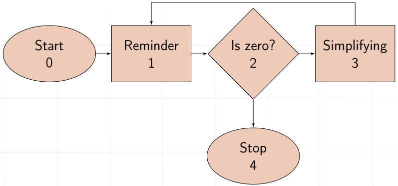
- What the computer program is?
  It is a kind of "control structure" defining the order and kind of operations performed on (input) data.
- Each algorithm is finite.
- Transition between instructions (states) depends on a current state and on the value(s) of certain data items.



Simplifying control — the thesis

- One of the things crucial to simplifying control is the finiteness of an algorithm's text.
- The processor can be in one of only finitely many locations in that text, and hence we can make do with a rather primitive mechanism, containing some kind of gearbox that can be in one of finitely many positions, or states.
- If we think of the states of the gearbox as encoding locations in the algorithm, then moving around in the algorithm can be modelled simply by changing states.

Example



State	m	n	r
0	10	18	—
1	10	18	10
2	10	18	10
3	18	10	10
1	18	10	8
2	18	10	8
3	10	8	8
1	10	8	2
2	10	8	2
3	8	2	2
1	8	2	0
2	8	2	0
4	8	2	0

The Turing Machine

A Turing machine M consists of

1. a (finite) set of **states**,
2. a (finite) **alphabet** of symbols,
3. an infinite **tape** with its marked-off squares and
4. a sensing-and-writing **head** that can travel along the tape, one square at a time.
5. a state transition **diagram**, sometimes called simply a transition diagram, containing the instructions that cause changes to take place at each step (the heart of the machine).

The Turing Machine

Formal description

The Turing machine, according to Hopcroft and Ullman is 7-tuple:

$$M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$$

where:

- Q is a finite set of states,
- Γ is an alphabet (finite set of symbols)
- $b \in \Gamma$ is an empty symbol
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is a set of input symbols
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a "transition diagram", (L standing for "left" and R standing for "right").
- $q_0 \in Q$ is an initial state.
- $F \subseteq Q$ is a set of "halting states."

The Turing Machine

A transition diagram

- A transition diagram can be viewed as a directed graph whose nodes represent the states. We use rounded rectangles (**rountangles** in the sequel) for states.

The Turing Machine

A transition diagram

- A transition diagram can be viewed as a directed graph whose nodes represent the states. We use rounded rectangles (**rountangles** in the sequel) for states.
- An edge leading from state s to state t is called a **transition**, and is labelled with a code of the form $a/b, L$ or $a/b, R$, where a and b are symbols.

The Turing Machine

A transition diagram

- A transition diagram can be viewed as a directed graph whose nodes represent the states. We use rounded rectangles (**rountangles** in the sequel) for states.
- An edge leading from state s to state t is called a **transition**, and is labelled with a code of the form $a/b, L$ or $a/b, R$, where a and b are symbols.
- The a part of the label is called the transition's **trigger**, and it denotes the letter read from the tape.

The Turing Machine

A transition diagram

- A transition diagram can be viewed as a directed graph whose nodes represent the states. We use rounded rectangles (**rountangles** in the sequel) for states.
- An edge leading from state s to state t is called a **transition**, and is labelled with a code of the form $a/b, L$ or $a/b, R$, where a and b are symbols.
- The a part of the label is called the transition's **trigger**, and it denotes the letter read from the tape.
- The b part is the **action**, and denotes the letter written on the tape.

The Turing Machine

A transition diagram

- A transition diagram can be viewed as a directed graph whose nodes represent the states. We use rounded rectangles (**rountangles** in the sequel) for states.
- An edge leading from state s to state t is called a **transition**, and is labelled with a code of the form $a/b, L$ or $a/b, R$, where a and b are symbols.
- The a part of the label is called the transition's **trigger**, and it denotes the letter read from the tape.
- The b part is the **action**, and denotes the letter written on the tape.
- Finally, the L and R part provides the direction to move, with L standing for "left" and R for "right." The precise meaning of a transition from s to t labelled with $a/b, L$ is as follows (the case for $a/b, R$ is similar):

The Turing Machine

A transition diagram

- A transition diagram can be viewed as a directed graph whose nodes represent the states. We use rounded rectangles (**rountangles** in the sequel) for states.
- An edge leading from state s to state t is called a **transition**, and is labelled with a code of the form $a/b, L$ or $a/b, R$, where a and b are symbols.
- The a part of the label is called the transition's **trigger**, and it denotes the letter read from the tape.
- The b part is the **action**, and denotes the letter written on the tape.
- Finally, the L and R part provides the direction to move, with L standing for "left" and R for "right." The precise meaning of a transition from s to t labelled with $a/b, L$ is as follows (the case for $a/b, R$ is similar):

  > *During its operation, whenever the Turing machine is in state s, and a is the symbol sensed at that moment by the head, the machine will erase the symbol a, writing b in its place, will move one square to the left, and will enter state t.*

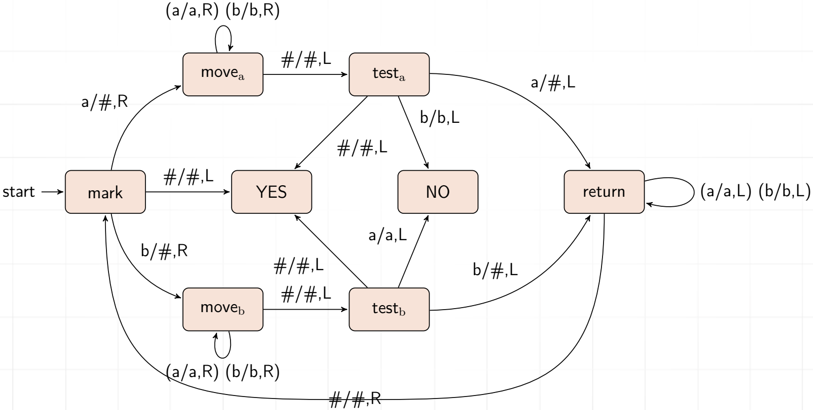
The Turing Machine: The Example

- **The Alphabet** — three symbols: **a**, **b** i # (means "empty" or "nothing")
- **Data**

...	#	#	a	b	b	a	#	#	...

The Turing Machine: The Example

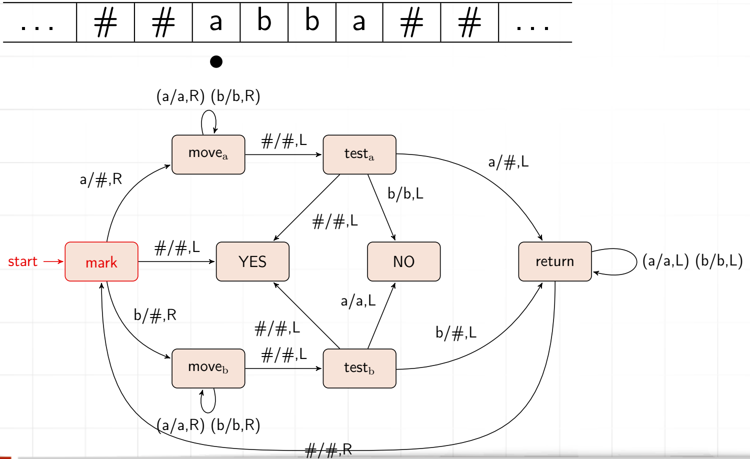
The transition diagram



Halting-states are described as “YES” and “NO.”

The Turing Machine

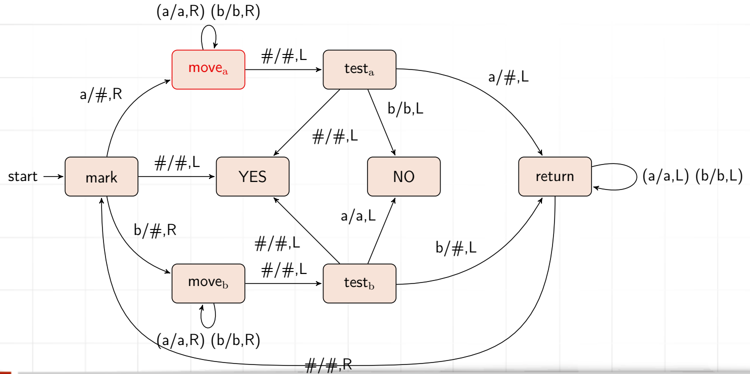
Example 1



The Turing Machine

Example 1

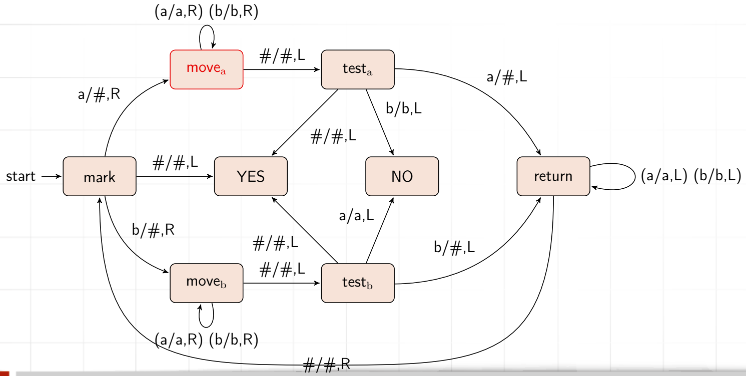
...	#	#	#	b	b	a	#	#	...



The Turing Machine

Example 1

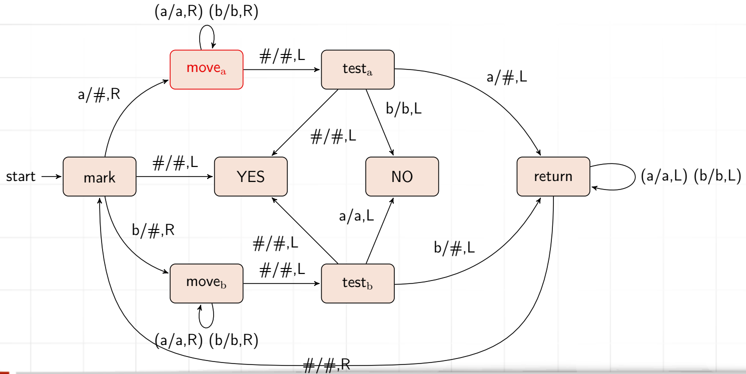
...	#	#	#	b	b	a	#	#	...



The Turing Machine

Example 1

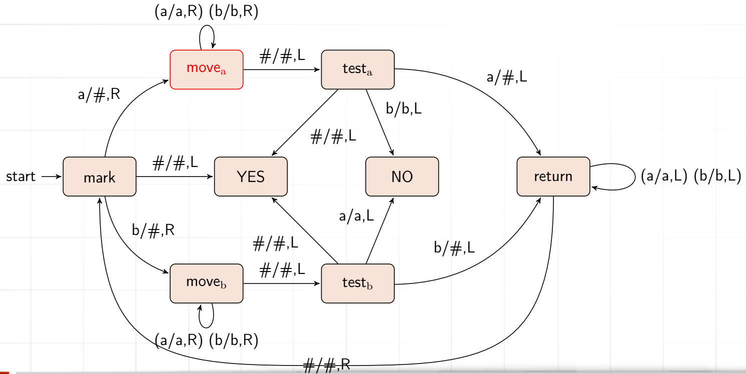
...	#	#	#	b	b	a	#	#	...



The Turing Machine

Example 1

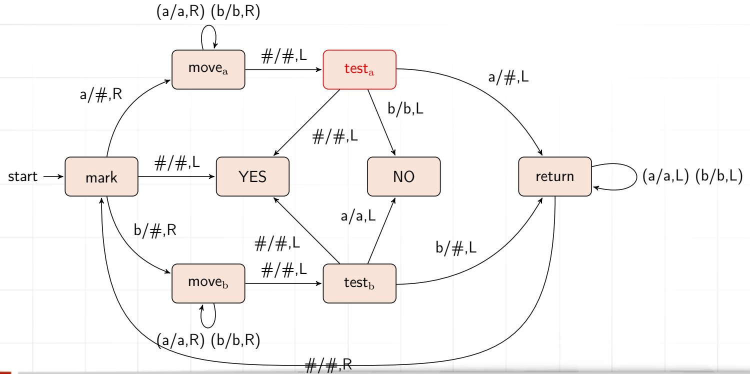
...	#	#	#	b	b	a	#	#	...



The Turing Machine

Example 1

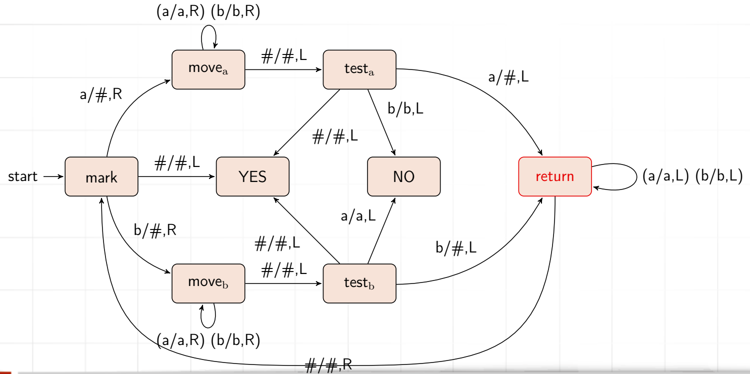
...	#	#	#	b	b	a	#	#	...



The Turing Machine

Example 1

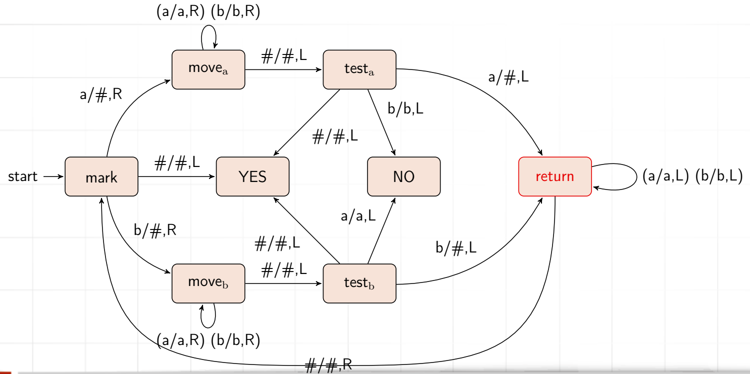
...	#	#	#	b	b	#	#	#	...



The Turing Machine

Example 1

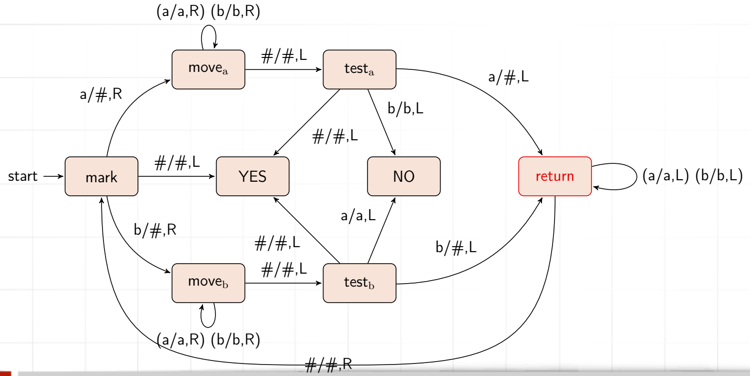
...	#	#	#	b	b	#	#	#	...



The Turing Machine

Example 1

...	#	#	#	b	b	#	#	#	...

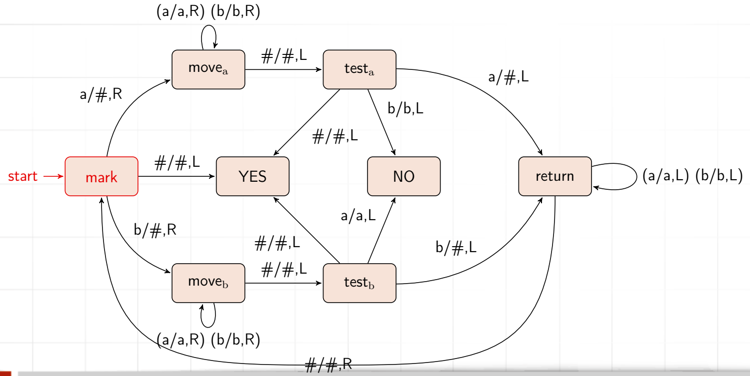


The Turing Machine

Example 1

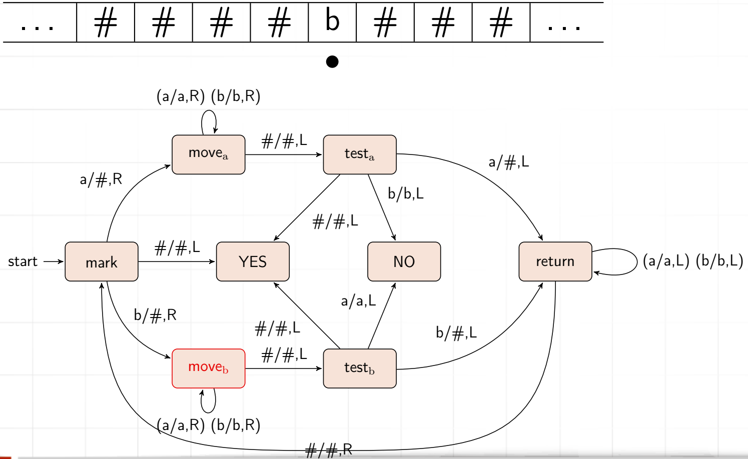
. . .	#	#	#	b	b	#	#	#	. . .

●



The Turing Machine

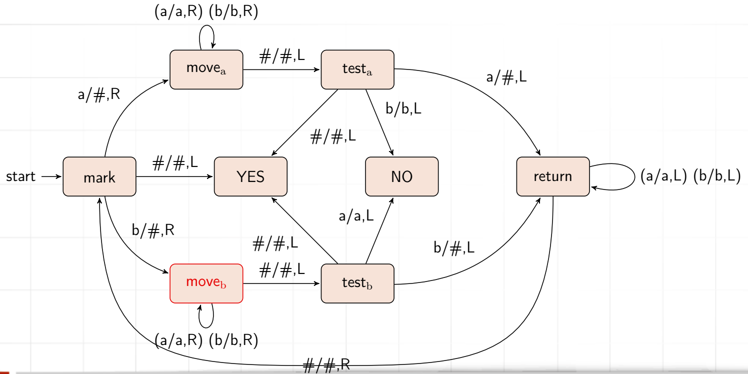
Example 1



The Turing Machine

Example 1

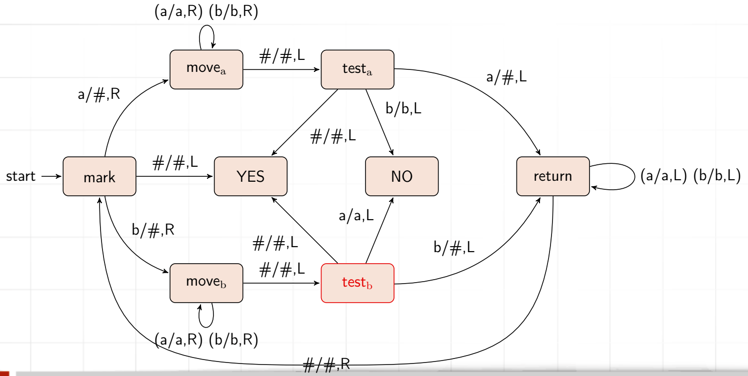
...	#	#	#	#	b	#	#	#	...



The Turing Machine

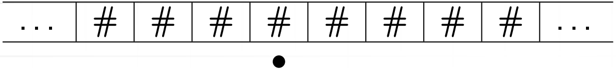
Example 1

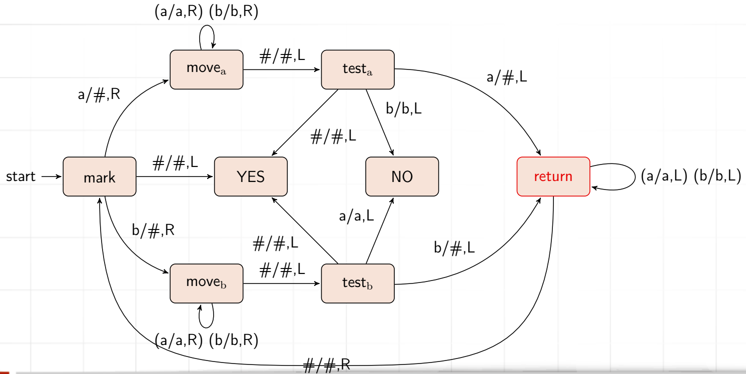
...	#	#	#	#	b	#	#	#	...



The Turing Machine

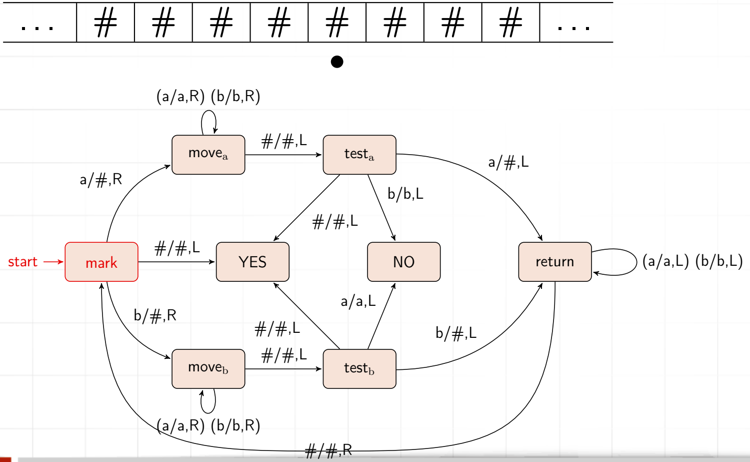
Example 1





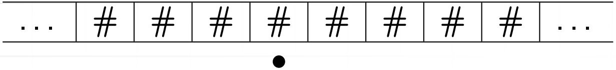
The Turing Machine

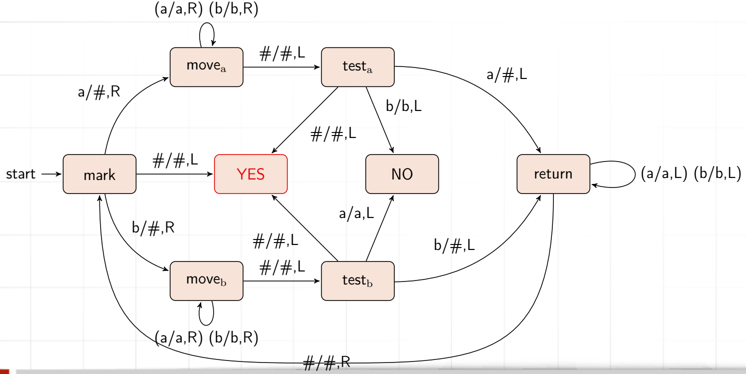
Example 1



The Turing Machine

Example 1

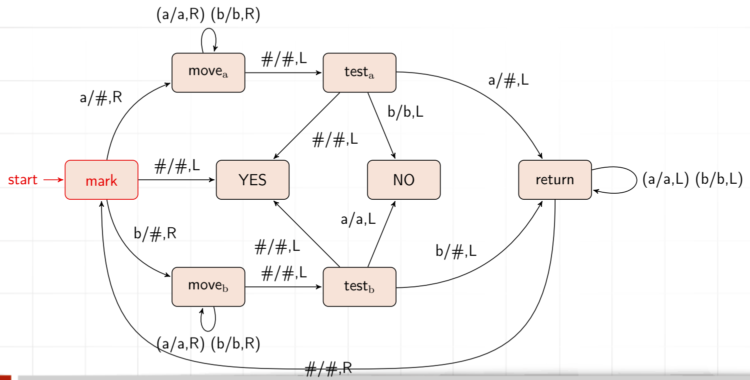




The Turing Machine

Example 2

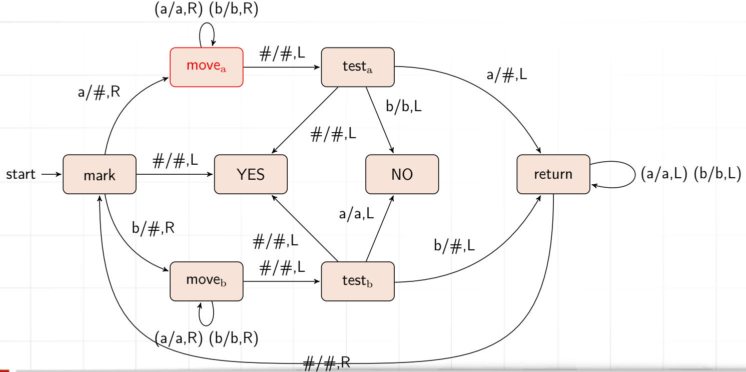
...	#	#	a	b	a	b	#	#	...



The Turing Machine

Example 2

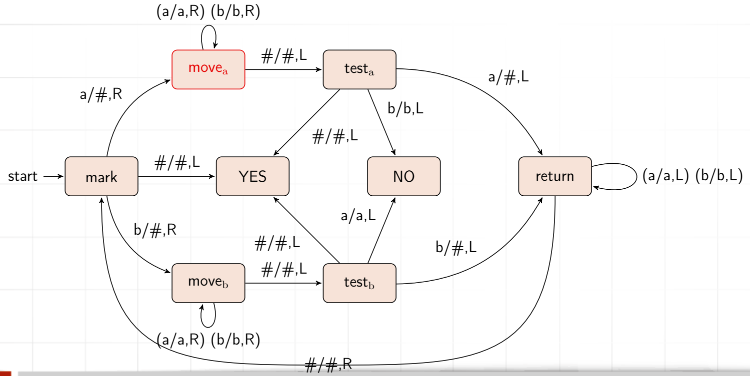
...	#	#	#	b	a	b	#	#	...



The Turing Machine

Example 2

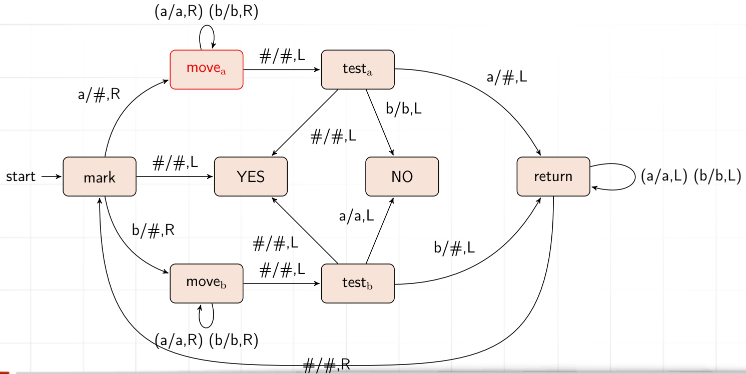
...	#	#	#	b	a	b	#	#	...



The Turing Machine

Example 2

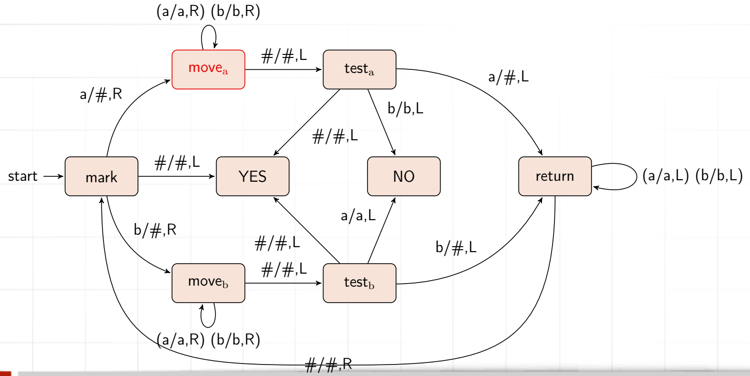
...	#	#	#	b	a	b	#	#	...



The Turing Machine

Example 2

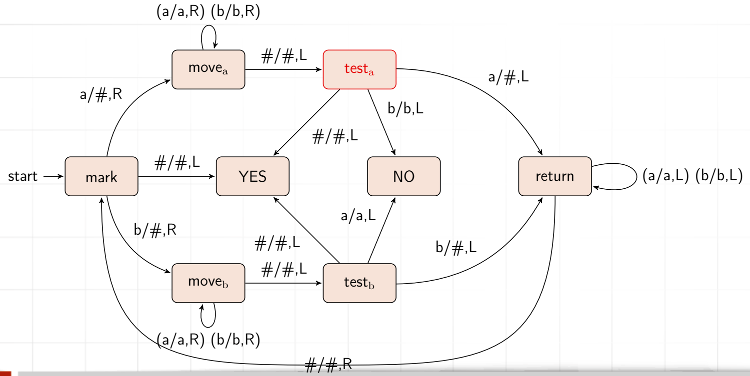
...	#	#	#	b	a	b	#	#	...



The Turing Machine

Example 2

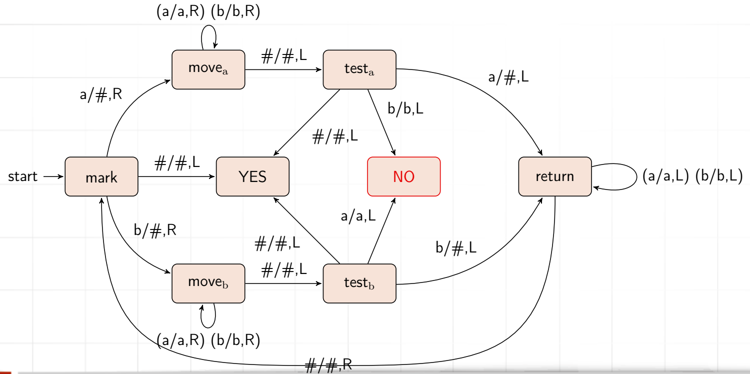
...	#	#	#	b	a	b	#	#	...



The Turing Machine

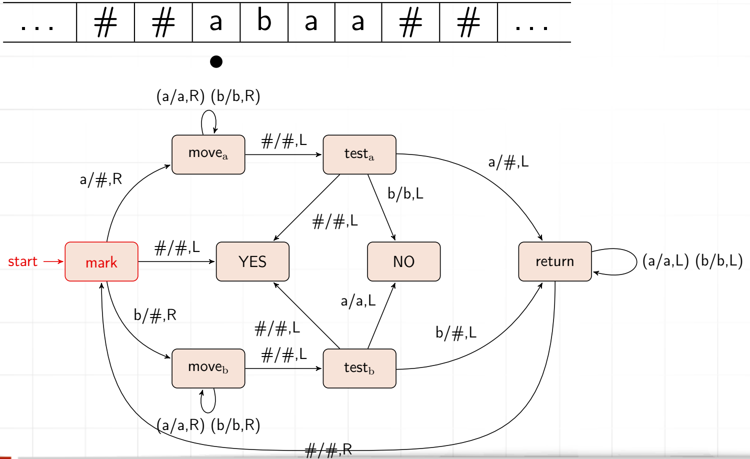
Example 2

...	#	#	#	b	a	b	#	#	...



The Turing Machine

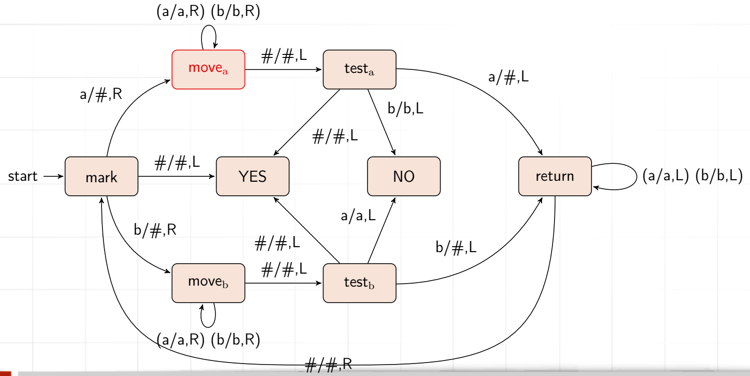
Example 3



The Turing Machine

Example 3

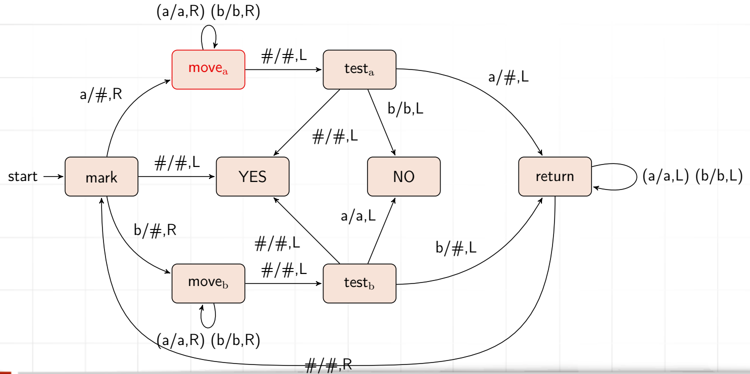
...	#	#	#	b	a	a	#	#	...



The Turing Machine

Example 3

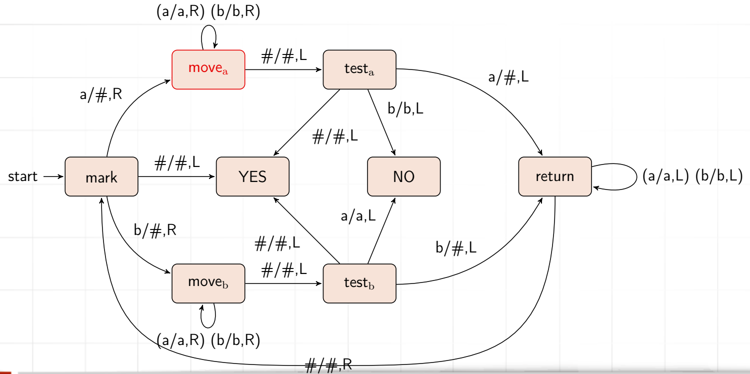
...	#	#	#	b	a	a	#	#	...



The Turing Machine

Example 3

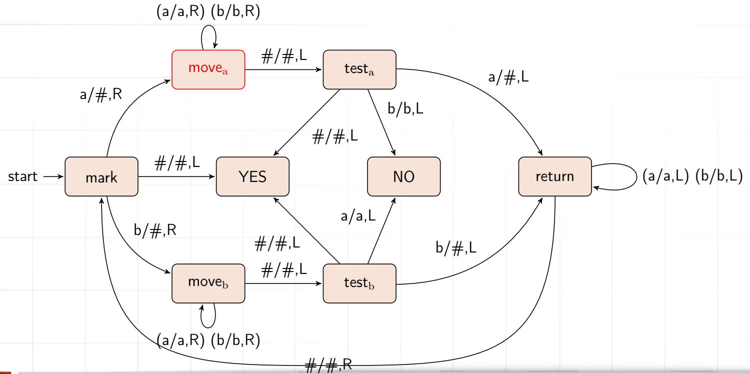
...	#	#	#	b	a	a	#	#	...



The Turing Machine

Example 3

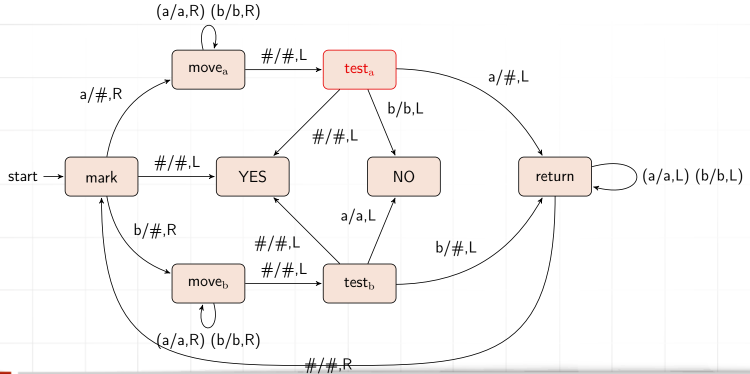
...	#	#	#	b	a	a	#	#	...



The Turing Machine

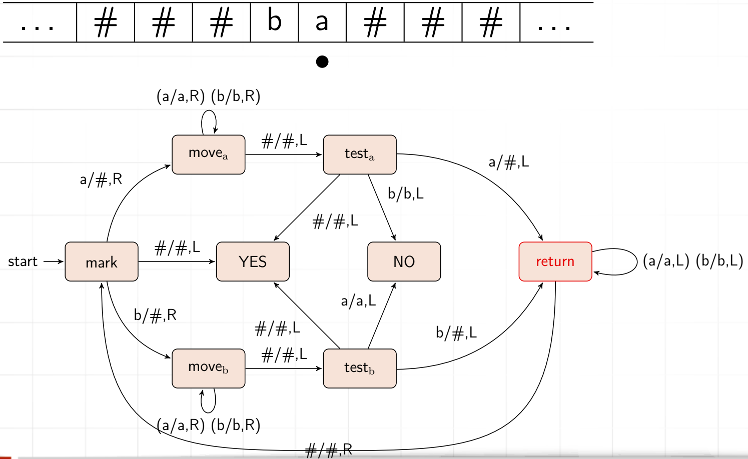
Example 3

...	#	#	#	b	a	a	#	#	...



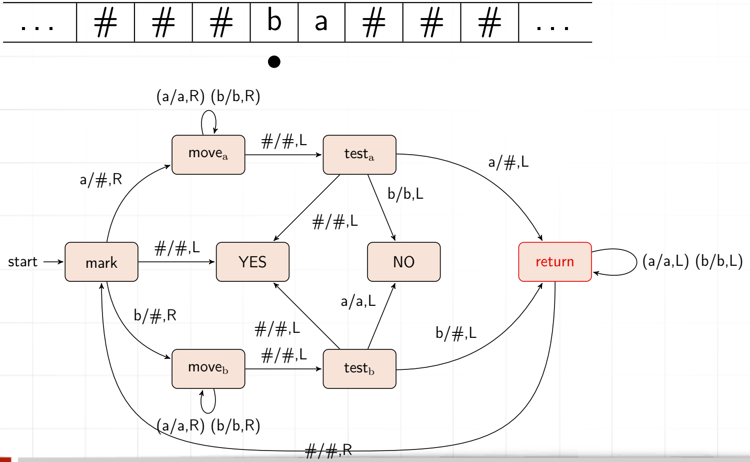
The Turing Machine

Example 3



The Turing Machine

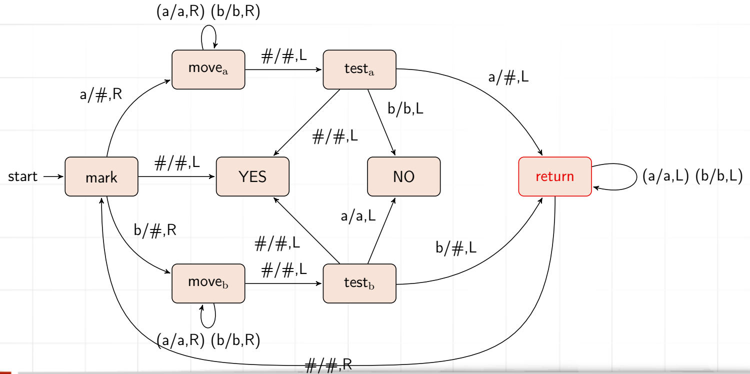
Example 3



The Turing Machine

Example 3

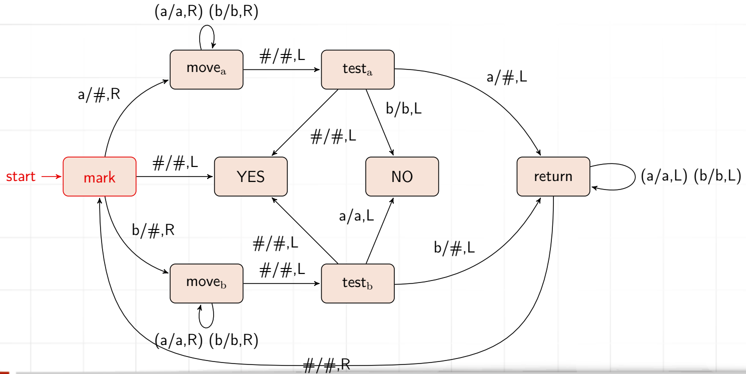
...	#	#	#	b	a	#	#	#	...



The Turing Machine

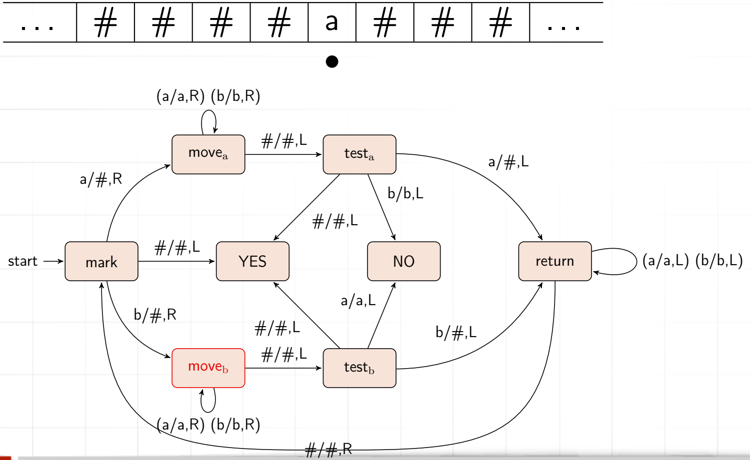
Example 3

...	#	#	#	b	a	#	#	#	...



The Turing Machine

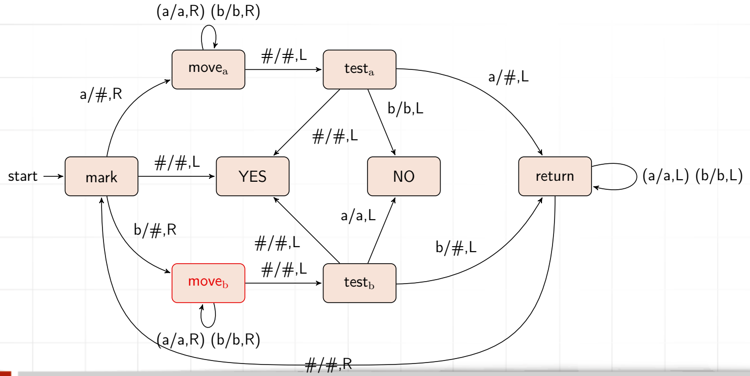
Example 3



The Turing Machine

Example 3

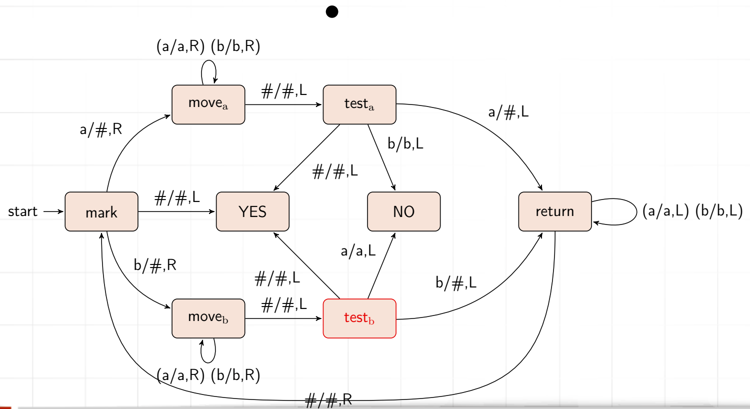
...	#	#	#	#	a	#	#	#	...



The Turing Machine

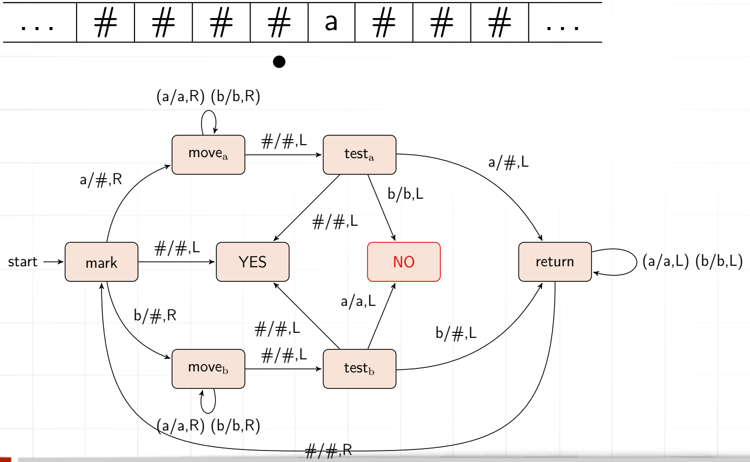
Example 3

...	#	#	#	#	a	#	#	#	...



The Turing Machine

Example 3



The Turing Machine

- A Turing Machine can be viewed as a computer with a single fixed program.
- The software is the transition diagram
- The hardware consists of the tape and head, as well as the (implicit) mechanism that actually moves through the transition diagram changing states and controlling the head's reading, writing, erasing, and moving activities.

The Turing Machine

Variants

- Turing machines with a tape that is infinite to the right only, the input appearing justified to the left, and the machine being constrained never to attempt to move "off" the leftmost square.

The Turing Machine

Variants

- Turing machines with a tape that is infinite to the right only, the input appearing justified to the left, and the machine being constrained never to attempt to move "off" the leftmost square.
- Two tapes: one for the input and one for the output.

The Turing Machine

Variants

- Turing machines with a tape that is infinite to the right only, the input appearing justified to the left, and the machine being constrained never to attempt to move "off" the leftmost square.
- Two tapes: one for the input and one for the output.
- Turing Machine with many tapes, each with its own read/write head, in such a way that the transitions are based on the entire set of symbols seen simultaneously by the heads.

The Turing Machine

Variants

- Turing machines with a tape that is infinite to the right only, the input appearing justified to the left, and the machine being constrained never to attempt to move "off" the leftmost square.
- Two tapes: one for the input and one for the output.
- Turing Machine with many tapes, each with its own read/write head, in such a way that the transitions are based on the entire set of symbols seen simultaneously by the heads.
- Two-dimensional tape giving rise to four, not two, possible moving directions.

The Turing Machine

Variants

- Turing machines with a tape that is infinite to the right only, the input appearing justified to the left, and the machine being constrained never to attempt to move "off" the leftmost square.
- Two tapes: one for the input and one for the output.
- Turing Machine with many tapes, each with its own read/write head, in such a way that the transitions are based on the entire set of symbols seen simultaneously by the heads.
- Two-dimensional tape giving rise to four, not two, possible moving directions.
- Non-deterministic Turing machines. The idea is to allow many transitions with the same trigger to emanate from a state. The machine then has a choice of which transition to take.

The Turing Machine

Variants

- Turing machines with a tape that is infinite to the right only, the input appearing justified to the left, and the machine being constrained never to attempt to move "off" the leftmost square.
- Two tapes: one for the input and one for the output.
- Turing Machine with many tapes, each with its own read/write head, in such a way that the transitions are based on the entire set of symbols seen simultaneously by the heads.
- Two-dimensional tape giving rise to four, not two, possible moving directions.
- Non-deterministic Turing machines. The idea is to allow many transitions with the same trigger to emanate from a state. The machine then has a choice of which transition to take.
- It can be (not-easily) proved that all this machines are equivalent (this meant that each of this machines can simulate any other).

The Turing Machine

1. Turing machine has only a finite number of states.
2. Programming must not be easy (try to build a machine multiplying two numbers!)
3. Its action is likely to be very slow, and the manual simulation is quite tedious.
4. But what really Turing machine can do?

- What indeed can be done with Turing machines, for whatever cost?

Turing Machine

The Church/Turing Thesis

- What indeed can be done with Turing machines, for whatever cost?
- Which algorithmic problems can be solved by an appropriately programmed Turing machine?

Turing Machine

The Church/Turing Thesis

- What indeed can be done with Turing machines, for whatever cost?
- Which algorithmic problems can be solved by an appropriately programmed Turing machine?
- Turing machines are capable of solving any effectively solvable algorithmic problem!

Turing Machine

The Church/Turing Thesis

- Any algorithmic problem for which we can find an algorithm that can be programmed in some programming language, any language, running on some computer, any computer, even one that has not been built yet but can be built, and even one that will require unbounded amounts of time and memory space for ever-larger inputs, is also solvable by a Turing machine.

Turing Machine

The Church/Turing Thesis

- Any algorithmic problem for which we can find an algorithm that can be programmed in some programming language, any language, running on some computer, any computer, even one that has not been built yet but can be built, and even one that will require unbounded amounts of time and memory space for ever-larger inputs, is also solvable by a Turing machine.

This statement is one version of the so-called Church/Turing (CT) thesis.

Turing Machine

The Church/Turing Thesis

- Any algorithmic problem for which we can find an algorithm that can be programmed in some programming language, any language, running on some computer, any computer, even one that has not been built yet but can be built, and even one that will require unbounded amounts of time and memory space for ever-larger inputs, is also solvable by a Turing machine.

This statement is one version of the so-called Church/Turing (CT) thesis.
It can be proved that after mentioned computing mechanical machine designed by Babbage (analytical engine) is equivalent to Turing Machine.

Church/Turing Thesis implications

1. It is important to realize that the CT thesis is a **thesis**, not a **theorem**, and hence cannot be proved in the mathematical sense of the word.
2. The CT thesis implies that the most powerful super-computer, with the most sophisticated array of programming languages, interpreters, compilers, assemblers, and what have you, is no more powerful than a home computer with its simplistic programming language.
3. All programming languages are equivalent (in the sense that every problem, which can be programmed in one language can be programmed in any other language as well. . .)

Counter Programs

Another Very Primitive Model

- A counter program can manipulate non-negative integers stored in variables.

Counter Programs

Another Very Primitive Model

- A counter program can manipulate non-negative integers stored in variables.
- The model, or language, allows just three types of elementary operations on variables, interpreted in the standard way (where, by convention, $Y - 1$ is defined to be 0 if Y is already 0):

The variables are called counters because the limited operations enable them, in essence, only to count.

Counter Programs

Another Very Primitive Model

- A counter program can manipulate non-negative integers stored in variables.
- The model, or language, allows just three types of elementary operations on variables, interpreted in the standard way (where, by convention, $Y - 1$ is defined to be 0 if Y is already 0):
  1. $X \leftarrow 0$,

The variables are called counters because the limited operations enable them, in essence, only to count.

Counter Programs

Another Very Primitive Model

- A counter program can manipulate non-negative integers stored in variables.
- The model, or language, allows just three types of elementary operations on variables, interpreted in the standard way (where, by convention, $Y - 1$ is defined to be 0 if Y is already 0):
  1. $X \leftarrow 0$,
  2. $X \leftarrow Y + 1$,

  The variables are called counters because the limited operations enable them, in essence, only to count.

Counter Programs

Another Very Primitive Model

- A counter program can manipulate non-negative integers stored in variables.
- The model, or language, allows just three types of elementary operations on variables, interpreted in the standard way (where, by convention, $Y - 1$ is defined to be 0 if Y is already 0):
  1. $X \leftarrow 0$,
  2. $X \leftarrow Y + 1$,
  3. $X \leftarrow Y - 1$

  The variables are called counters because the limited operations enable them, in essence, only to count.

Counter Programs

Another Very Primitive Model

- A counter program can manipulate non-negative integers stored in variables.
- The model, or language, allows just three types of elementary operations on variables, interpreted in the standard way (where, by convention, $Y - 1$ is defined to be 0 if Y is already 0):
  1. $X \leftarrow 0$,
  2. $X \leftarrow Y + 1$,
  3. $X \leftarrow Y - 1$

  The variables are called counters because the limited operations enable them, in essence, only to count.
- The control structures of a counter program include simple sequencing and the conditional **goto** statement:
  if $X = 0$ **goto** G

Counter Programs

Another Very Primitive Model

- A counter program can manipulate non-negative integers stored in variables.
- The model, or language, allows just three types of elementary operations on variables, interpreted in the standard way (where, by convention, $Y - 1$ is defined to be 0 if Y is already 0):
  1. $X \leftarrow 0$,
  2. $X \leftarrow Y + 1$,
  3. $X \leftarrow Y - 1$

  The variables are called counters because the limited operations enable them, in essence, only to count.
- The control structures of a counter program include simple sequencing and the conditional **goto** statement:
  if $X = 0$ **goto** G

Counter Programs

Another Very Primitive Model

- A counter program can manipulate non-negative integers stored in variables.
- The model, or language, allows just three types of elementary operations on variables, interpreted in the standard way (where, by convention, $Y - 1$ is defined to be 0 if Y is already 0):
  1. $X \leftarrow 0$,
  2. $X \leftarrow Y + 1$,
  3. $X \leftarrow Y - 1$

  The variables are called counters because the limited operations enable them, in essence, only to count.
- The control structures of a counter program include simple sequencing and the conditional **goto** statement:
  if $X = 0$ **goto** G

Let us note that this model is much closer to the computer model proposed by von Neumann than a Turing machine!

Counter Program

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

input data X and Y, output data Z; non-existent label L means end of the program.

Counter Programs

Example

$U \leftarrow 0$

$Z \leftarrow 0$

A: **if** $X = 0$ **goto** L

$X \leftarrow X - 1$

$V \leftarrow Y + 1$

$V \leftarrow V - 1$

B: **if** $V = 0$ **goto** A

$V \leftarrow V - 1$

$Z \leftarrow Z + 1$

if $U = 0$ **goto** B

X	Y	V	Z
2	3		

Counter Programs

Example

$U \leftarrow 0$

$Z \leftarrow 0$

A: **if** $X = 0$ **goto** L

$X \leftarrow X - 1$

$V \leftarrow Y + 1$

$V \leftarrow V - 1$

B: **if** $V = 0$ **goto** A

$V \leftarrow V - 1$

$Z \leftarrow Z + 1$

if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
2	3		0

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
2	3		0

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
1	3		0

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
1	3	4	0

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
1	3	3	0

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
1	3	3	0

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
1	3	2	0

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
1	3	2	1

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
1	3	2	1

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
1	3	2	1

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
1	3	1	1

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
1	3	1	2

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
1	3	1	2

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
1	3	1	2

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
1	3	0	2

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
1	3	0	3

Counter Programs

Example

$U \leftarrow 0$

$Z \leftarrow 0$

A: **if** $X = 0$ **goto** L

$X \leftarrow X - 1$

$V \leftarrow Y + 1$

$V \leftarrow V - 1$

B: **if** $V = 0$ **goto** A

$V \leftarrow V - 1$

$Z \leftarrow Z + 1$

if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
1	3	0	3

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
1	3	0	3

Counter Programs

Example

$U \leftarrow 0$

$Z \leftarrow 0$

A: **if** $X = 0$ **goto** L

$X \leftarrow X - 1$

$V \leftarrow Y + 1$

$V \leftarrow V - 1$

B: **if** $V = 0$ **goto** A

$V \leftarrow V - 1$

$Z \leftarrow Z + 1$

if $U = 0$ **goto** B

X	Y	V	Z
1	3	0	3

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	0	3

Counter Programs

Example

$U \leftarrow 0$

$Z \leftarrow 0$

A: **if** $X = 0$ **goto** L

$X \leftarrow X - 1$

$V \leftarrow Y + 1$

$V \leftarrow V - 1$

B: **if** $V = 0$ **goto** A

$V \leftarrow V - 1$

$Z \leftarrow Z + 1$

if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
0	3	4	3

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	3	3

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	3	3

Counter Programs

Example

$U \leftarrow 0$

$Z \leftarrow 0$

A: **if** $X = 0$ **goto** L

$X \leftarrow X - 1$

$V \leftarrow Y + 1$

$V \leftarrow V - 1$

B: **if** $V = 0$ **goto** A

$V \leftarrow V - 1$

$Z \leftarrow Z + 1$

if $U = 0$ **goto** B

X	Y	V	Z
0	3	2	3

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
0	3	2	4

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	2	4

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
0	3	2	4

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	1	4

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	1	5

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	1	5

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	1	5

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
0	3	0	5

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	0	6

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	0	6

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	0	6

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	Y	V	Z
0	3	0	6

Counter Programs

Example

$U \leftarrow 0$
$Z \leftarrow 0$
A: **if** $X = 0$ **goto** L
$X \leftarrow X - 1$
$V \leftarrow Y + 1$
$V \leftarrow V - 1$
B: **if** $V = 0$ **goto** A
$V \leftarrow V - 1$
$Z \leftarrow Z + 1$
if $U = 0$ **goto** B

X	**Y**	**V**	**Z**
0	3	0	6

End

Counter program

It can be proved that the counter program can simulate Turing machine and that the Turing machine can simulate the counter program.