cse303
ELEMENTS OF THE THEORY OF COMPUTATION

Professor Anita Wasilewska

LECTURE 13

CHAPTER 4
TURING MACHINES

1. The definition of Turing machine
2. Computing with Turing machines
3. Extensions of Turing machines

CHAPTER 5
UNDECIDABILITY

1. The Church-Turing thesis
2. Universal Turing machines
3. Undecidable problems about Turing machines

CHAPTER 4
TURING MACHINES

1. The definition of Turing machine
2. Computing with Turing machines
3. Extensions of Turing machines

The definition of Turing machine

Short History

The **Turing machine** was invented in 1936 by Alan Turing

From Wikipedia:

" . **Alan Mathison Turing** (23 June 1912 7 June 1954), was a British mathematician, logician, cryptanalyst, and computer scientist. He was highly influential in the development of computer science, giving a formalization of the concepts of "algorithm" and "computation" with the Turing machine, which can be considered a model of a general purpose computer. He is widely considered to be the father of computer]science and artificial intelligence"

Short History

Automata Theory is a theoretical branch of computer science developed by mathematicians during the 20th Century

It deals with the logic of **computation** with respect to simple machines, referred to as **automata**

Through **automata**, computer scientists are able to understand how machines compute functions and solve problems and what it means for a function to be defined as **computable** or for a question to be described as **decidable**

Short History

The first description of **finite automata** was presented in 1943 by Warren McCulloch and Walter Pitts, two neurophysiologists

Their theory was **generalized** to much more powerful **machines** by G.H. Mealy and E.F. Moore in separate papers, published in 1955 - 56

Turing machine is the most general and the most powerful **automata**

Short History

Context-free **Grammars** were developed after Chomsky work published 1957

From Wikipedia:

" . **Avram Noam Chomsky** (born December 7, 1928) is an American linguist, philosopher, cognitive scientist, logician, political commentator and activist. Sometimes is described as the "father of modern linguistics"

Chomsky has spent most of his career at the Massachusetts Institute of Technology (MIT), where he is currently Professor Emeritus, and has authored over 100 books. He has been described as a prominent cultural figure, and was voted the "world's top public intellectual" in a 2005 poll."

The definition of Turing machine

Finite and **Pushdown** automata can't be regarded as truly general models of computers, or computations because they can't recognize even such simple languages as

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

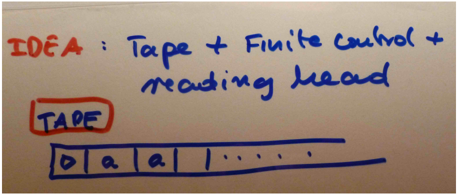
Turing machines can recognize those- and more complicated languages and much more!

Turing machines are not automata, but are similar in their design

The definition of Turing machine

The **main idea** of the TMachine is similar to the Finite and Pushdown automata;

We also have a tape, a finite control module and a reading head

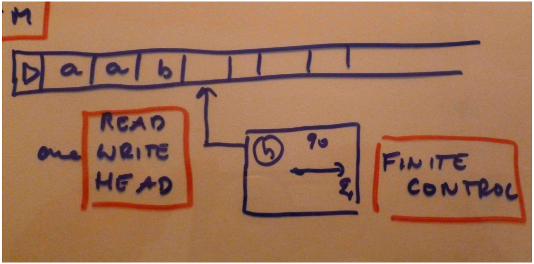


We assume that the tape has a special symbol ▷ for the leftmost end of the tape

We can have the Turing Machines models without it

The definition of Turing machine

Turing Machine consists of a Finite Control unit, a tape, and a read/write head that moves in both directions



The read/write head **reads** symbols from the tape and is also used to **change** symbols on the tape

T Machine can move the head one square at the time

We visit only a finite number of squares during a finite computation

The definition of Turing machine

Finite Control at each step performs 2 functions dependent on a current state and a symbol scanned by the **reading head**

Function 1: puts the **FC** unit in a new state

Function 2: performs either;

(a) writes a symbol in the tape square currently scanned, replacing the one already there; or

(b) moves the read/write head one tape square to the left or right

The tape has left end, but it extends indefinitely to the right

The definition of Turing machine

To prevent the machine from moving its head off the left end of the tape we assume that the tape has always a special symbol $\triangleright$ for the leftmost end of the tape

We also assume that all our **T Machines** are so designed that, whenever the **head reads** a symbol $\triangleright$, it immediately **moves** to the right

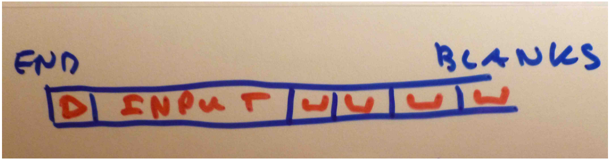
We use the distinct symbols $\leftarrow$ and $\rightarrow$ to denote movement of the **head** to the left and the right, respectively

The symbols $\leftarrow$ and $\rightarrow$ are not members of any **alphabet** we consider

The symbol $\triangleright$ is **never erased**

The definition of Turing machine

Turing Machine is supplied with input by writing the input string on the tape immediately after the symbol $\triangleright$
The rest of the tape initially contains **blank** symbols, denoted by $\sqcup$



T Machine is free to **alter** its input
It also can **write** on the unlimited **blank portion** of the tape to the right

Turing Machine Mathematical Model

Definition

A **Turing Machine** is a quintuple

$$TM = (K, \Sigma, \delta, s, H)$$

where

K is a finite set of **states**

Σ as an **alphabet**

Σ contains a **blank** symbol $\sqcup$ and a **left end** symbol $\triangleright$

Σ **does not** contain symbols $\leftarrow$ and $\rightarrow$

$s \in K$ is the **initial state**

$H \subseteq K$ is the set of **halting states**

We usually use different symbols for K, Σ, i.e. we have that $K \cap \Sigma = \emptyset$

Turing Machine Mathematical Model

Turing Machine components continue

δ is a **transition function**

$$\delta : \ (K - H) \times \Sigma \longrightarrow K \times (\Sigma \cup \{\rightarrow, \leftarrow\})$$

such that the following conditions hold

1. If $\delta(q, \triangleright) = (p, b)$ then $b = \rightarrow$

2. If $\delta(q, a) = (p, b)$ then $b \neq \triangleright$

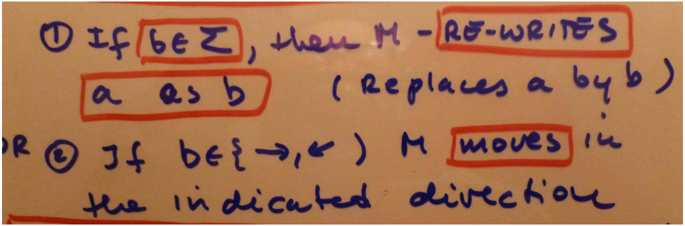
Observe that δ is a function, so **Turing Machine** is always **deterministic**

Operation of Turing machine

If $q \in K - H, \;\; a \in \Sigma$ and

$$\delta(q,\ a) = (p, b)$$

then TM in state q scanning a on the tape will enter state p and



Operation of Turing machine

TM **stops** only when enters a **halting** state $h \in H$

Observe that δ is not defined for $h \in H$

TM has also two **extra requirements** -

the two **extra conditions** in the definition of δ

Condition 1. If TM sees $\triangleright$ (end of the tape) then TM must move right:

$$\text{If} \quad \delta(q, \triangleright) = (p, b) \quad \text{then} \quad b = \rightarrow$$

It means that symbol $\triangleright$ is never erased and TM never gets out of the tape

Operation of Turing machine

Condition 2. of the definition of δ is

$$\text{If} \quad \delta(q, a) = (p, b) \quad \text{then} \quad b \neq \triangleright$$

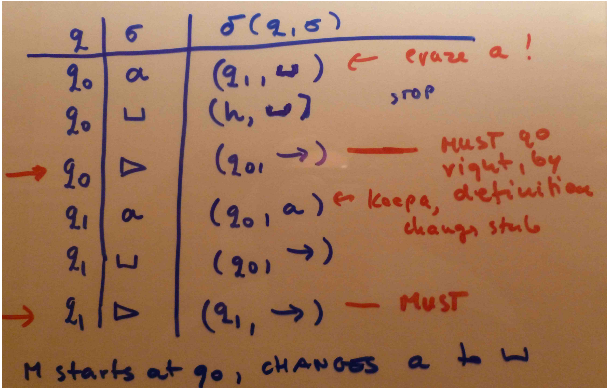
It says that TM **never** writes on $\triangleright$

Observe that the **conditions 1.** and **2.** guarantee that the symbol $\triangleright$ is well defined and acts a **protective barrier**

Turing Machine Examples

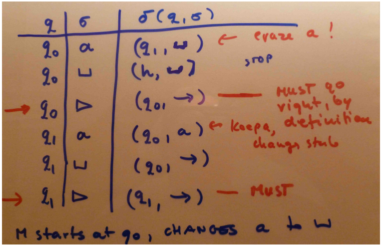
Example 1

Let $M = (K, \Sigma, \delta, s, H)$ where $K = \{q_0, q_1, h\}$, $s = q_0$, $\Sigma = \{a, \sqcup, \triangleright\}$, $H = \{h\}$ and δ is given by the table



Examples

Operation of M - lets look at δ again



M starts at q_0, changes a to $\sqcup$ (erases a) and goes to q_1

When M in q_1 sees a - and goes to q_0 - and q_0 erases a

When M in q_0 sees $\sqcup$ - M **halts**

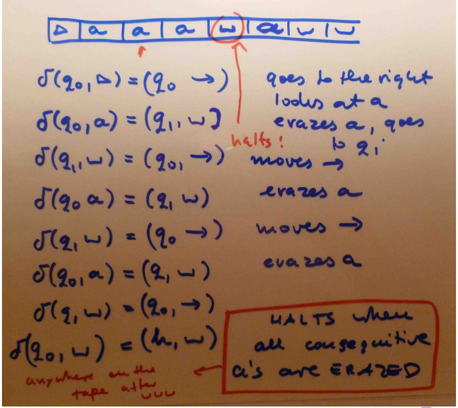
When M in q_1 sees $\sqcup$ - M goes to q_0 and **moves** right

Examples

Operation of M

Remark: assignment $\delta(q_1, a) = (q_0, a)$ is irrelevant because M never can be in state q_1 scanning a) if it started at q_0 - THIS is like a TRAP State; δ must be a function

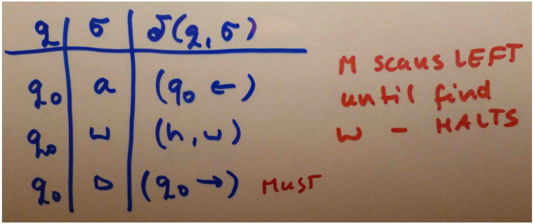
Here a computation of M



Turing Machine Examples

Example 2

Let $M = (K, \Sigma, \delta, s, H)$ where $K = \{q_0, h\}$, $s = q_0$, $\Sigma = \{a, \sqcup, \triangleright\}$, $H = \{h\}$ and δ is given by the table



If every tape square from the **head** position to the left contains an a the M will go to the left end of the tape and then M **indefinitely** goes between the left end and the square to its right

Operation of M may **never stop**

Formal Definition

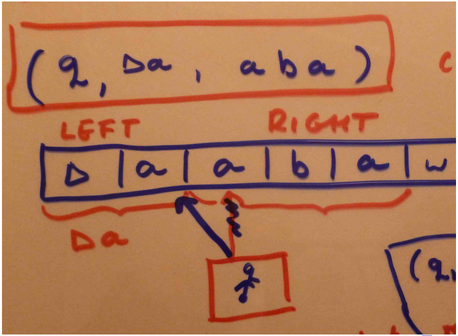
Formal definition of operation of Turing Machine is similar to the one for FA and PD automata

We define first a notion of a configuration

Configuration of M is any element

$$(q, \triangleright w, u) \in K \times \triangleright\Sigma^* \times (\Sigma^*(\Sigma - \{\sqcup\}) \cup \{e\})$$

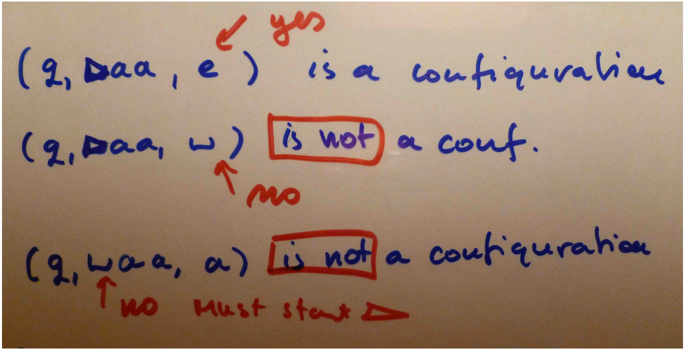
Picture



Configuration

Configuration/not Configuration examples

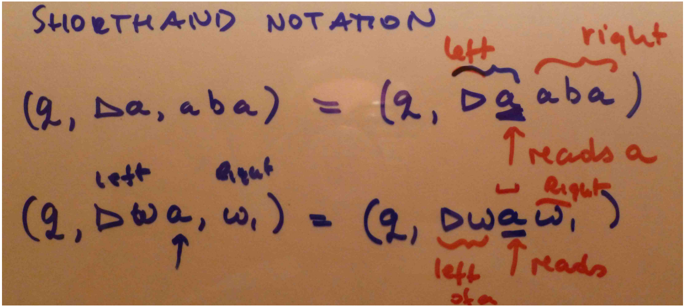
Picture



Configuration

Configuration shorthand notation

Picture



Halted Configuration is a configuration whose state components is in H

Transition Relation

Given a set S of **all configurations** of M

$$S \subseteq K \times \triangleright\Sigma^* \times (\Sigma^*(\Sigma - \{\sqcup\}) \cup \{e\})$$

The **transition relation** acts between two **configurations** and hence $\vdash_M$ is a certain binary relation defined on $S \times S$, i.e.

$$\vdash_M \subseteq (K \times \triangleright\Sigma^* \times (\Sigma^*(\Sigma - \{\sqcup\}) \cup \{e\}))^2$$

We write

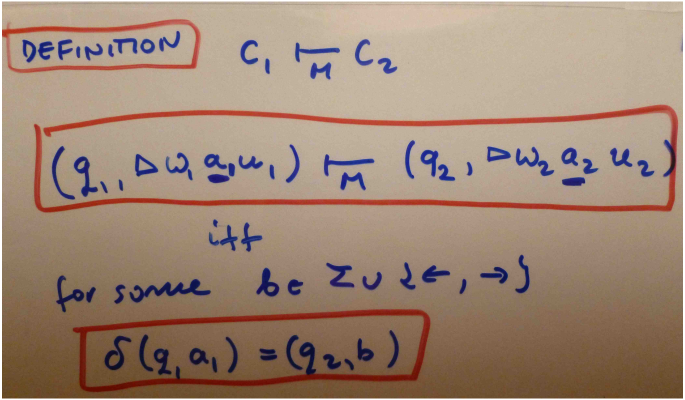
$$C_1 \vdash_M C_2$$

and say C_1 YIELDS C_2

Formal definition follows

Transition Relation

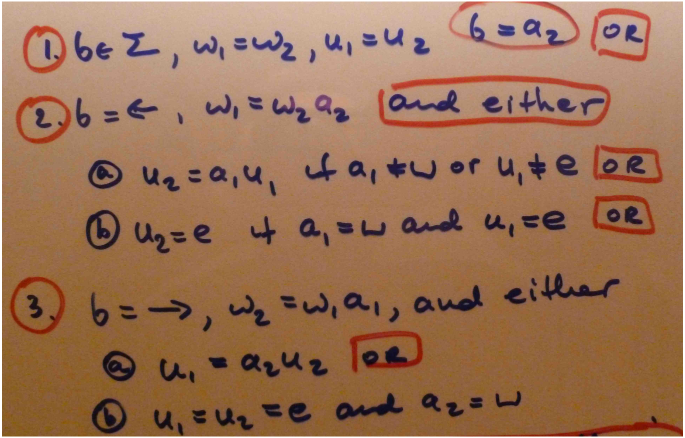
Definition of $C_1 \vdash_M C_2$



and **either**

Transition Relation

and **either**



Computation by TM

Given a **transition relation** $\vdash_M$

We **denote** as usual, its reflexive, transitive closure is denoted by $\vdash_M{}^*$ and

$$C_1 \vdash_M{}^* C_n$$

is a **computation** of the length n in M from C_1 to C_n

By definition of $\vdash_M{}^*$ we have that

$$C_1 \vdash_M{}^* C_n$$

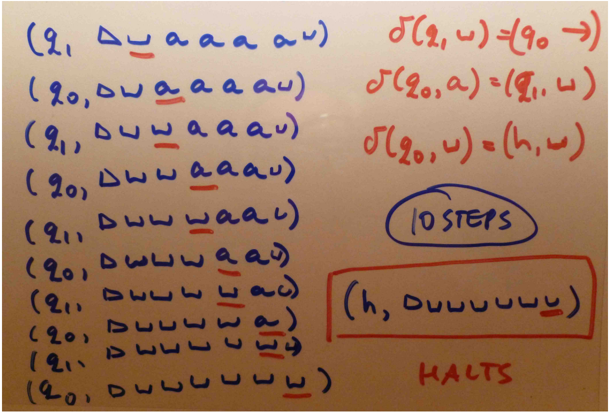
if and only of

$$C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n$$

Computation by TM

Let M be the Turing Machine from **Example 1** that scans the tape to the right **changing** a's to ⊔ until **finds** a blank ⊔ and then **halts**

Here is a computation of M of lengths 10



A Notation for Turing Machines

A Notation for Turing Machines

The **Turing machines** we have seen so far are extremely **simple** but their transition function is already complex and difficult to understand and interpret

We shall now adopt a **graphical representation** for Turing machines similar to the **diagrams** for finite automata

However, in this case the diagrams' nodes will be **not states**, but *themselves* **Turing machines**

A Notation for Turing Machines

We use a *hierarchical* notation, in which more and more complex machines are built from simpler materials

To this end we define a very simple repertoire of **basic machines**, together with **rules** for combining machines

We will be **building** machines by combining the basic machines, and then we shall further **combine** the combined machines to obtain more complex machines, and so on

Basic Machines

We fix the alphabet Σ and define the *symbol-writing* and *head moving* machines as follows

For each $a \in (\Sigma \cup \{\rightarrow, \leftarrow\}) - \{\triangleright\}$ we define a TM

$$M_a = (\{s, h\}, \ \Sigma, \ \delta, \ s, \ \{h\})$$

where for each $b \in \Sigma - \{\triangleright\}$

$\delta(s, b) = (h, a)$ and as always, $\delta(s, \triangleright) = (s, \rightarrow)$

Basic Machines

Given the TM machine

$$M_a = (\{s, h\},\ \ \Sigma,\ \ \delta,\ \ s,\ \ \{h\})$$

the **only thing** this machine **does** is to perform action of

writing a symbol a if $a \in \Sigma$

and M_a is called a **symbol-writing** machine,

moving to the direction indicated by a if $a \in \{\rightarrow, \leftarrow\}$

and M_a is called a **head-moving** machine,

and then M_a immediately **halts**

If **scanned** symbol is a $\triangleright$,

then the machine will dutifully **move** to the right

Basic Machines

Let M_a be a **symbol -writing** or **head-moving** machine

We adopt the following **notation**:

1. If $a \in \Sigma$, we write

$$a \quad \text{instead of} \quad M_a$$

for a**-writing** machine M_a

2. If $a \in \{\rightarrow, \leftarrow\}$, we write

L and R instead of $M_{\leftarrow}$ and $M_{\rightarrow}$, respectively

The Rules for Combining Machines

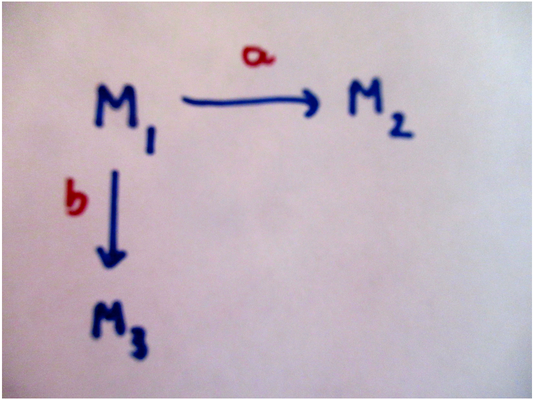
We combine the machines treating the individual machines like the **states** of finite automata
The machines may be connected to each other in the way that the **states** of finite automaton are connected **together**

But the connection from one machine to the other **does not** happen **until** the first machine **halts**
The other machine is then **started** from the initial state with the tape and head position as they were **left** by the first machine

Turing Machine Diagram

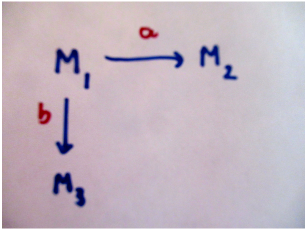
Given Turing Machines M_1, M_2, M_3

Here us a **diagram** of a machine $M = (K, \Sigma, \delta, s, H)$ composed of M_1, M_2, M_3



Turing Machine Diagrams

Given a diagram of of a machine $M = (K, \Sigma, \delta, H)$



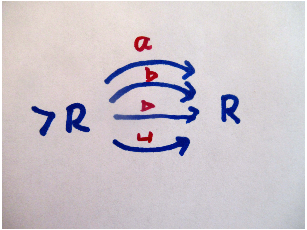
M **starts** at the initial state of M_1; **operates** as M_1 until M_1 **halts**; then

if the currently scanned symbol is an a **initiates** M_2, and **operates** as M_2; otherwise,

if the currently scanned symbol is a b, then **initiates** M_3, and operates as M_3

Turing Machine Diagrams

Here is a **diagram** of a machine M **consisting** of **two copies** of the basic machine $R = M_{\rightarrow}$



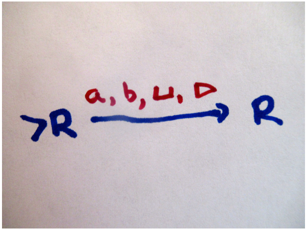
M **moves** its head right one square;

if that square contains an a, or a b, or a ▷, or a ⊔,

it **moves** its head one square to the right

Diagrams of Turing Machines

It is convenient to represent the machine M **consisting** of **two copies** of the R machine as follows



If an arrow is labelled by **all** symbols of the alphabet Σ of the machines, then the labels can be **omitted** and we denote the **diagram** as

$R \longrightarrow R$ or under this convention, as RR or even R^2

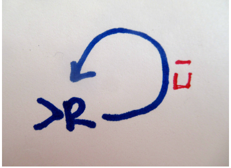
Diagrams of Turing Machines

Here are some more convenient **diagrams**

Let $a \in \Sigma$ be any symbol. We use a symbol $\overline{a}$ to say

"any symbol except a"

The **diagram**



represents a machine that **scans** the tape to the right **until** it finds a blank $\sqcup$. We denote it by $R_{\sqcup}$

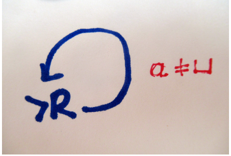
Diagrams of Turing Machines

Let $a \in \Sigma$ be any symbol

We write a symbol $a \neq \sqcup$ to denote a statement

"any symbol a *other than* $\sqcup$ "

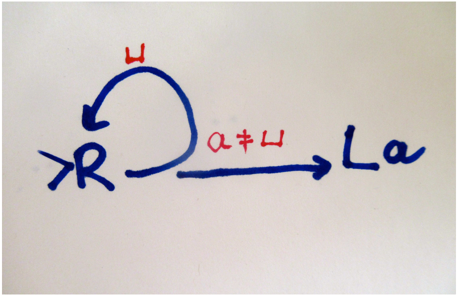
The **diagram**



is another representation of the machine $R_{\sqcup}$ that **scans** the tape to the right **until** it finds a blank $\sqcup$

Diagrams of Turing Machines

The **diagram**

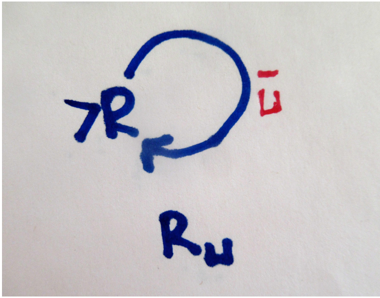


depicts a machine that **scans** to the right **until** it finds a nonblank square, then **copies** the symbol in that square **onto** the square immediately to the left of where it was **found**

Some Simple Special Machines

Here are machines to find marked or unmarked squares

The **diagram**

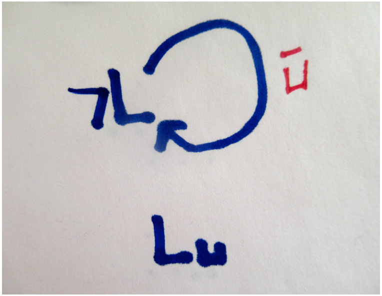


depicts a machine $R_\sqcup$ that **finds** the first blank square to the **right** of the currently scanned square

Some Simple Special Machines

Here are machines to find marked or unmarked squares

The **diagram**

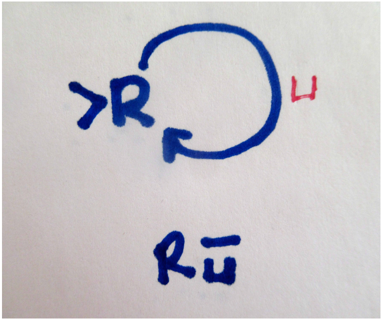


depicts a machine $L_{\sqcup}$ that **scans** the tape to the left **until** it **finds** a blank $\sqcup$

Some Simple Special Machines

Here are machines to find marked or unmarked squares

The **diagram**

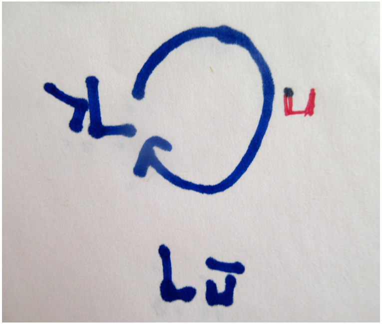


depicts a machine $R_{\overline{\sqcup}}$ that **finds** the first NONBLANK square to the **right** of the currently scanned square

Some Simple Special Machines

Here are machines to find marked or unmarked squares

The **diagram**



depicts a machine $L_{\overline{\sqcup}}$ that **finds** the first NONBLANK square to the **left** of the currently scanned square

CHAPTER 4
TURING MACHINES

1. The definition of Turing machine
2. Computing with Turing machines
3. Extensions of Turing machines

Computing with Turing machines

Computing with Turing machines

We introduced Turing Machines with the **goal** that they outperform, as **language acceptors**, all of automata we introduced and examined

To be able discuss this **goal** we have to define (and examine) how they are to be **used** to perform a task of **language recognition**

In order to do so we **need** to fix some conventions for **use** of Turing machines

Computing with Turing machines

We **adopt** the following policy for presenting **input** to Turing machines

1. The **input** string, with no blank symbols in it, is written to the right of the leftmost symbol $\triangleright$, **with** a blank on its left, and blanks to its right

2. The **head** is positioned at the tape square containing the blank between the $\triangleright$ and the **input**
3. Machine **starts** operation in its initial state

Computing with Turing machines

Given a **Turing machine**

$$M = (K,\ \Sigma,\ \delta,\ s,\ H)$$

and let

$$w \in (\Sigma - \{\sqcup,\ \triangleright)^*\}$$

The **initial configuration** of M on **input** word w is

$$(s,\ \triangleright\underline{\sqcup}w)$$

Computing with Turing machines

Consider a **Turing Machine** $M = (K, \Sigma, \delta, s, H)$

for $H = \{y, n\}$ where y denotes **accepting** configuration

n denotes **rejecting** configuration

M **accepts** a word $w \in (\Sigma - \{\sqcup, \triangleright\})^*$ if and only if the **initial** configuration $(s, \triangleright \underline{\sqcup} w)$ on input word w yields an **accepting** configuration

M **rejects** a word $w \in (\Sigma - \{\sqcup, \triangleright\})^*$ if and only if the **initial** configuration $(s, \triangleright \underline{\sqcup} w)$ on input word w yields an **rejecting** configuration

Computing with Turing machines

Given a Turing machine $M = (K, \Sigma, \delta, s, H)$ for $H = \{y, n\}$

The alphabet

$$\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$$

is called an **input alphabet** of M

By fixing Σ_0 to be subset of $\Sigma - \{\sqcup, \triangleright\}$ we allow our Turing machines to use extra symbols during their computation, besides those appearing in their inputs

Recursive Languages

Given an **input alphabet** $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$ of M

Definition

M **decides** a language $L \subseteq \Sigma_0{}^*$ if for any word $w \in L$ the following condition holds

If $w \in L$ then M **accepts** M **decides** w;

and if $w \notin L$ then M **rejects** w

Definition

The language $L \subseteq \Sigma_0{}^*$ is **recursive** if there is a Turing machine M that **decides** it

Recursive Languages

Observe that M **decides** a language if, when started with input w, it always **halts**, and does so in a halt state that is a **correct response** to the input:

y if $w \in L$,

n if $w \notin L$

Notice that no guarantees sre given about what happens if the input to M contains blank or the left end symbol

Recursive Languages

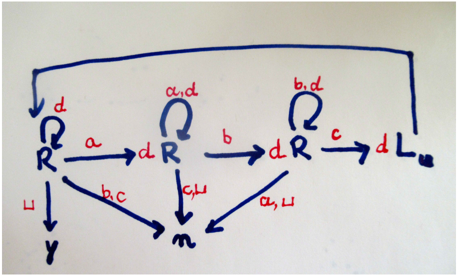
Theorem

The **not context-free** language

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

is recursive

Proof Here is a **diagram** of a Turing machine M that **decides** L



Recursive Functions

Recursive Functions

Let f be any function from ${\Sigma_0}^*$ to ${\Sigma_0}^*$

Definition

A Turing machine M **computes** function f if for **all** words $w \in {\Sigma_0}^*$ it eventually **halts** on input w,
and when it does **halt**, its tape **contains** the string

$$\rhd \sqcup f(w)$$

Definition

A function f is called **recursive**, if **there is** a Turing machine M that **computes** f

Recursively Enumerable Languages

Definition

A Turing machine M **semidecides** a language $L \subseteq {\Sigma_0}^*$ if and only if for **any** word $w \in {\Sigma_0}^*$ the following is true:

$w \in L$ if and only if M **halts** on input w

Definition

A language $L \subseteq {\Sigma_0}^*$ is **recursively enumerable** if **there is** a Turing machine M that **semidecides** it

Recursively Enumerable Languages

Observe that a Turing machine M that **semidecides** a language L when presented with **input** $w \in L$, is required to **halt eventually**

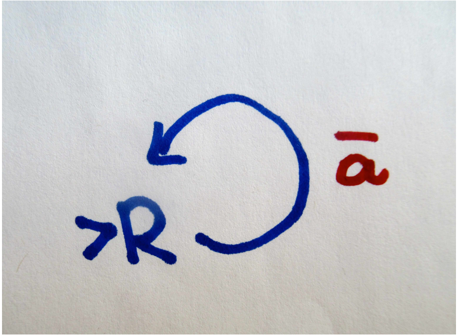
We **do not** care precisely **which** halting configuration it reaches, as long as it does eventually **arrive** at a halting configuration

Theorem
The language

$$L = \{w \in \{a.b\}^* : \ w \text{ contains at least one } a \ \}$$

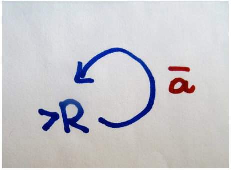
recursively enumerable

Proof L is **semidecided** by M defined by the following diagram



Recursively Enumerable Languages

The machine M



when **started** in **initial configuration** on **input** w, i.e. in configuration $(s, \triangleright\underline{\sqcup}w)$ for some $w \in \{a.b\}^*$ simply scans right until an a is **found** and then **halts**

If **no** a is **found**, the machine goes on forever onto the blanks that follow its input, **never halting**

Two important Theorems

Theorem 1

If a language is **recursive** then it is **recursively enumerable**

Theorem 2

If $L \subseteq \Sigma_0{}^*$ is a **recursive** language, then its complement $\Sigma_0{}^* - L$ is also **recursive**

CHAPTER 4
TURING MACHINES

1. The definition of Turing machine
2. Computing with Turing machines
3. Extensions of Turing machines

Extensions of Turing machines

Multiple tapes

ORIGINAL Turing Machine 1936 : two way infinite tape

Two dimensional tape

Random access machines

Non-deterministic machines

Theorem

All models of Turing Machine are computationally equivalent to the **standard** Turing machine M

ChurchTuring Thesis

A Turing machine that is able to **simulate** any other Turing machine is called a **universal** Turing machine UTM, or simply a universal machine

A more mathematically-oriented **definition** with a similar ”universal” nature was introduced by **Alonzo Church**

Church work on lambda calculus intertwined with Turing's in a statement known in the a **formal** theory of computation as **ChurchTuring Thesis**

ChurchTuring Thesis

The **ChurchTuring thesis** states that Turing machines indeed **capture** the informal notion of **effective method** in logic and mathematics, and provide a **precise definition** of an algorithm or ”mechanical procedure”

Studying the **abstract properties** of the Turing machines yields many insights into **computer science** and complexity theory