Chapter 4 Turing Machine

Elements Of The Theory Of Computation
Zhejiang University/CS/Course/Xiaogang Jin
E-Mail: xiaogangj@cise.zju.edu.cn

Handicapped machines

- **DFA limitations**
  - Tape head moves only one direction
  - Tape is read-only
  - Tape length is a constant

- **PDA limitations**
  - Tape head moves only one direction
  - Tape is read-only, but stack is writable
  - Stack has only LIFO(last-in, first-out) access
  - Tape length is constant, but stack is not bounded.

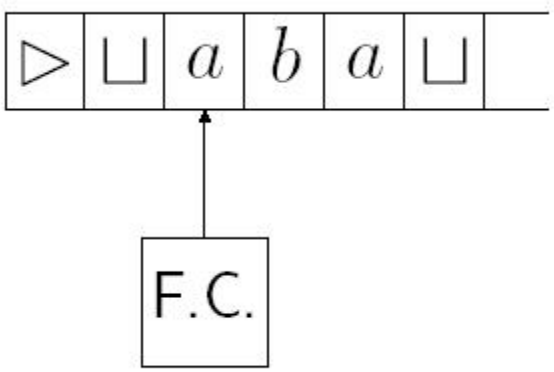
- **What about**
  - Writable, 2-way tape?
  - Random-access 'stack?

Computability

- **Computible**
  - Equivalence of DFAs
  - String membership in a regular language

- **Uncomputible**
  - Equivalence of CFGs
  - Emptiness of the complement of a CFL (emptiness of CFL is computible)

4.1 The Definition of Turing Machines



- Head can both read and write, and move in both directions
- Tape has unbounded length.
- ␣ is blank symbol. In practice, all but a finite number of tape squares are blank.

Definition: A Turing Machine is a quintuple $(K, \sum, \delta, s, H)$, where

- K is a finite set of states
- $\sum$ is an alphabet
  - containing $\sqcup$(blank symbol) and $\triangleright$(left end)
  - not containing the symbols $\leftarrow$ and $\rightarrow$
- $s \in K$ is the initial state
- $H \subseteq K$ is the set of halting states
- δ: $(K - H) \times \sum \rightarrow K \times (\sum \cup \{\leftarrow, \rightarrow\})$ be the transition function

a) $\forall q \in K - H$ if $\delta(q, \triangleright) = (p, b)$, then $b = \rightarrow$.

b) $\forall q \in K - H$ and $a \in \sum$, if $\delta(q, a) = (p, b)$, then $b \neq \triangleright$.

Example:

Consider the Turing Machine $M = (K, \sum, \delta, s, \{h\})$, where

$K = \{q_0, q_1, h\}$
$\sum = \{a, \sqcup, \triangleright\}$
$s = q_0$
and δ is given by the right table.

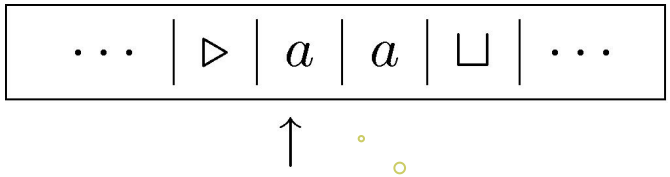
q	σ	$\delta(q, \sigma)$
q_0	a	$(q_1, \sqcup)$
q_0	$\sqcup$	$(h, \sqcup)$
q_0	$\triangleright$	$(q_0, \rightarrow)$
q_1	a	(q_0, a)
q_1	$\sqcup$	$(q_0, \rightarrow)$
q_1	$\triangleright$	$(q_1, \rightarrow)$

Example:

Consider the Turing Machine $M = (K, \sum, \delta, s, H)$, where

$K = \{q_0, h\}$
$\sum = \{a, \sqcup, \triangleright\}$
$s = q_0$
$H = \{h\}$
and δ is given by the right table.

q	σ	$\delta(q, \sigma)$
q_0	a	$(q_0, \leftarrow)$
q_0	$\sqcup$	$(h, \sqcup)$
q_0	$\triangleright$	$(q_0, \rightarrow)$



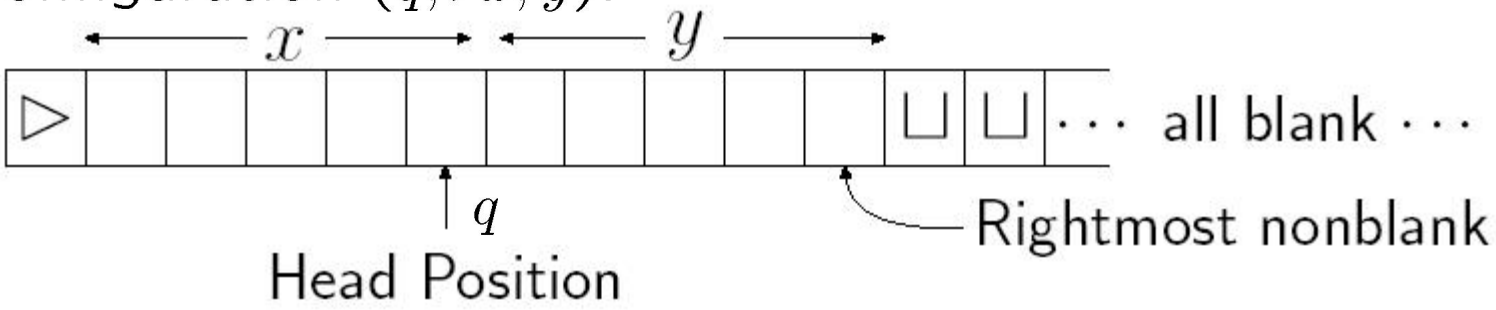
Never stop!

(The machine goes into a loop if no ⊔ can[illegible]nd.)

Turing Machines Configuration

Definition: A configuration of a TM $M = (K, \sum, \delta, s, H)$ is a member of $K \times \triangleright \sum^* \times (\sum^* (\sum - \{\sqcup\}) \cup \{e\})$.

Configuration $(q, \triangleright x, y)$:



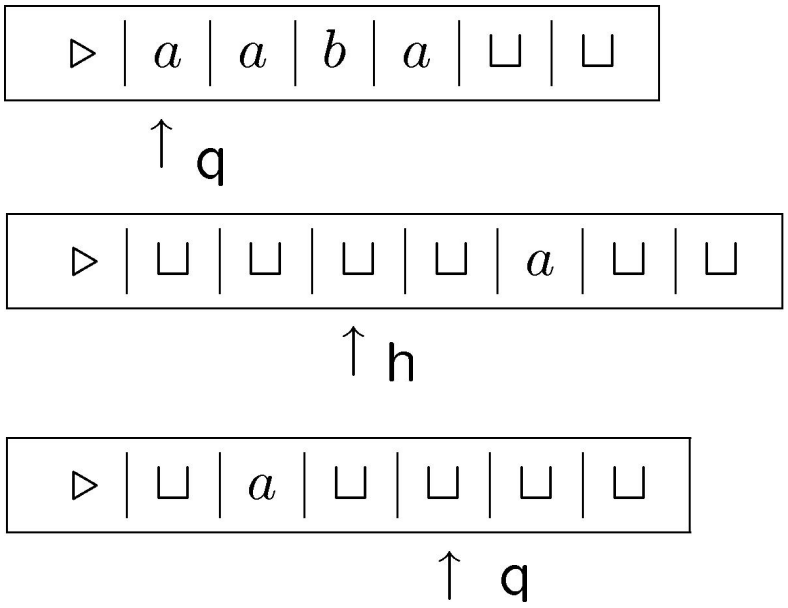
Remark:

- A configuration whose state component is in H will be called halted configuration.

- A simplified notation of configuration:

$$(q, wa, u) \Rightarrow (q, w\underline{a}u)$$

Example of configuration:



$(q, \triangleright a, aba)$
or $(q, \triangleright \underline{a} aba)$

$(h, \triangleright \sqcup \sqcup \sqcup, \sqcup a)$
or $(h, \triangleright \sqcup \sqcup \underline{\sqcup} \sqcup a)$

$(q, \triangleright \sqcup a \sqcup \sqcup, e)$
or $(q, \triangleright \sqcup a \sqcup \underline{\sqcup} e)$

Turing Machines Computation

Definition: Let $M = (K, \sum, \delta, s, H)$ be a TM and consider two configurations of M $(q_1, w_1\underline{a_1}u_1)$ and $(q_2, w_2\underline{a_2}u_2)$ where $a_1, a_2 \in \sum$. Then

$$(q_1, w_1\underline{a_1}u_1) \vdash_M (q_2, w_2\underline{a_2}u_2)$$

iff, for some $b \in \sum \cup \{\leftarrow, \rightarrow\}$, $\delta(q_1, a_1) = (q_2, b)$, either

1) $b \in \sum$, $w_1 = w_2$, and $u_1 = u_2$, and $a_2 = b$ or

2) $b = \leftarrow$, $w_1 = w_2 a_2$ and either

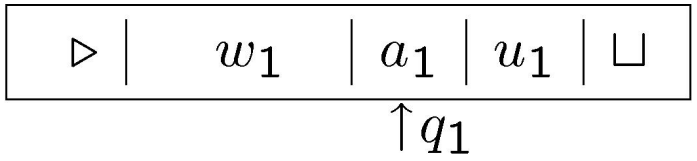
   (a) $u_2 = a_1 u_1$, if $a_1 \neq \sqcup$ or $u_1 \neq e$, or

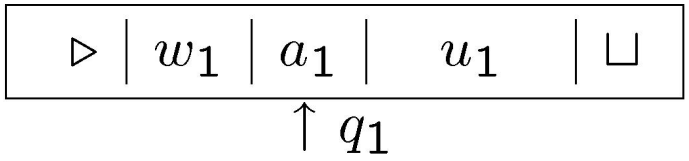
   (b) $u_2 = e$, if $a_1 = \sqcup$ and $u_1 = e$;

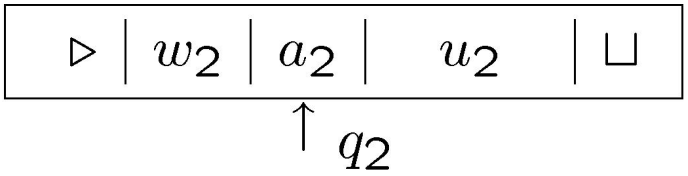
3) $b = \rightarrow$, $w_2 = w_1 a_1$ and either

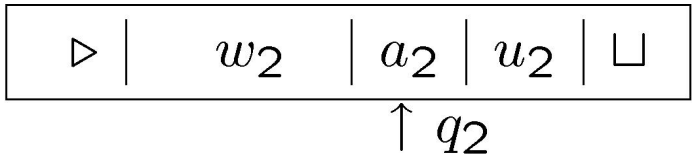
   (a) $u_1 = a_2 u_2$, if $a2 \neq \sqcup$ or $u_2 \neq e$, or

   (b) $u_2 = e$, if $a_2 = \sqcup$ and $u_2 = e$.









2) $b = \leftarrow$

3) $b = \rightarrow$

Remark:

- For any Turing Machine M, let $\vdash_M^*$ be the Reflexive, transitive closure of $\vdash_M$.
Configuration C_1 yields configuration C_2 if $C_1 \vdash_M^* C_2$.

- A **computation** by M is a sequence of configuration C_0, C_1, $\cdots$, C_n, for some $n \geq 0$ such that

$$C_0 \vdash_M C_1 \vdash_M \cdots \vdash_M C_n$$

we say that the computation is of length n or that it has n steps, denoted by $C_0 \vdash_M^n C_n$.

A Notation for Turing Machines

- **Basic Machines**

  Symbol-writing and hand-moving Machines:

  $\sum$ –a fixed alphabet, $a \in \sum \cup \{\leftarrow, \rightarrow\} - \{\triangleright\}$, define a Turing Machine

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

  where for each $b \in \sum - \{\triangleright\}, \delta(s, b) = (h, a)$.

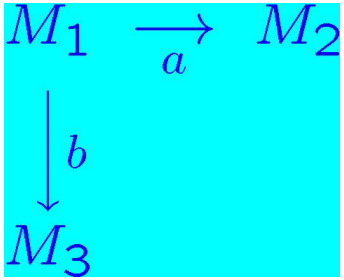
  Naturally, $\delta(s, \triangleright) = (s, \rightarrow)$.

  Abbreviate:

  a instead of M_a, L instead of $M_{\leftarrow}$, and R instead of $M_{\rightarrow}$.

- **The rules for combining Machines**

M_1, M_2 and M_3 are Turing Machines, the Machine displayed in figure operates as follows:



Start in the initial state of M_1, operate as M_1 would operate until M_1 halt; then if the currently scanned symbol is an a, initiate M_2 and then operate as M_2 would operate; otherwise if the currently scanned symbol is a b, initiate M_3 and then operate as M_3 would operate.

Formally, $M_i = (K_i, \sum, \delta_i, s_i, H_i)$, $(i = 1, 2, 3)$ be TMs.
The combined machine would be $M = (K, \sum, \delta, s, H)$, where

$K = K_1 \cup K_2 \cup K_3$

$s = s_1$

$H = H_2 \cup H_3$

For each $\sigma \in \sum$ and $q \in K - H, \delta(q, \sigma)$ is defined as follows:

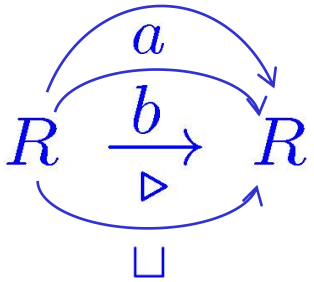
(a) If $q \in K_1 - H_1$, then $\delta(q, \sigma) = \delta_1(q, \sigma)$.

(b) If $q \in K_2 - H_2$, then $\delta(q, \sigma) = \delta_2(q, \sigma)$.

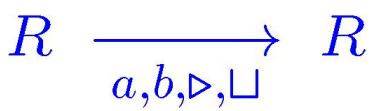
(c) If $q \in K_3 - H_3$, then $\delta(q, \sigma) = \delta_3(q, \sigma)$.

(d) If $q \in H_1$, then $\delta(q, \sigma) = s_2$ if $\sigma = a$;$\delta(q, \sigma) = s_3$ if $\sigma = b$; otherwise $\delta(q, \sigma) \in H$.

Example:



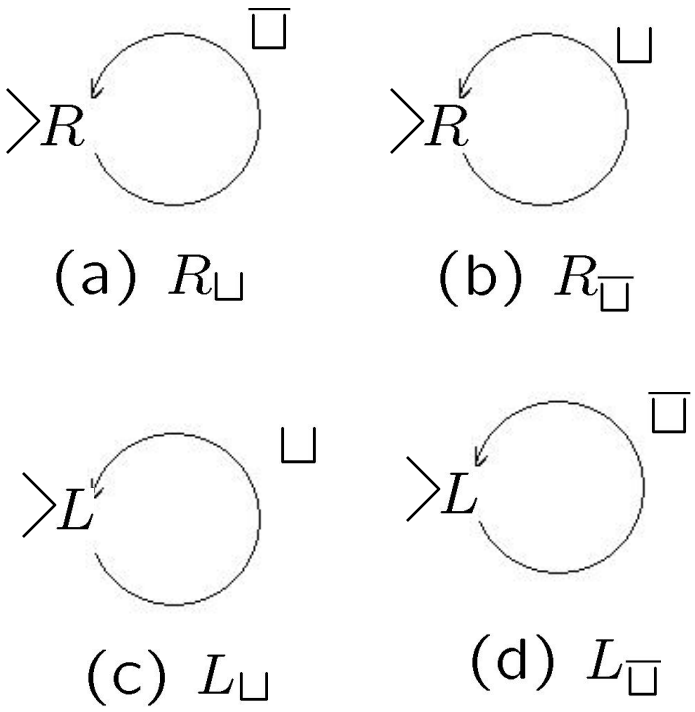
(a)



(b)

- *The machine represented by the diagram (a) moves its head right one square; then if that square contains* a*, or* b*, or* $\triangleright$*, or* $\sqcup$*, it moves its head one square further to the right.*
- *The machine can be simply represented by (b).*
- *If* $\sum = \{a, b, \triangleright, \sqcup\}$*, the machine become simply* R^2*.*

Example:



(a) $R_{\sqcup}$ (b) $R_{\overline{\sqcup}}$

(c) $L_{\sqcup}$ (d) $L_{\overline{\sqcup}}$

(a) $R_{\sqcup}$, finds the first blank square to the right of the currently scanned square.

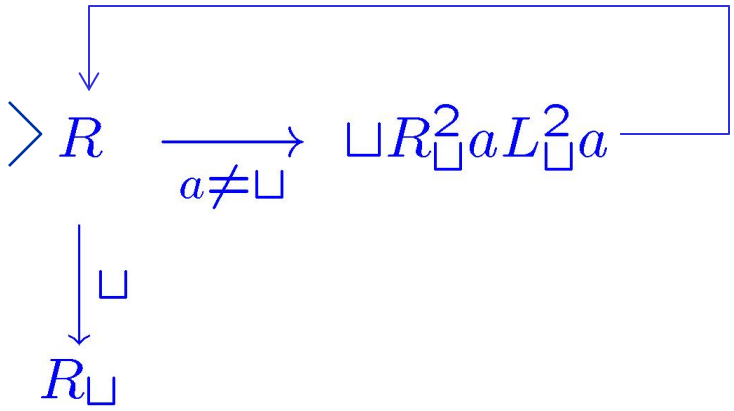
(b) $R_{\overline{\sqcup}}$, finds the first nonblank square to the right of the currently scanned square.

(c) $L_{\sqcup}$, finds the first blank square to the left of the currently scanned square.

(d) $L_{\overline{\sqcup}}$, finds the first nonblank square to the left of the currently scanned square.

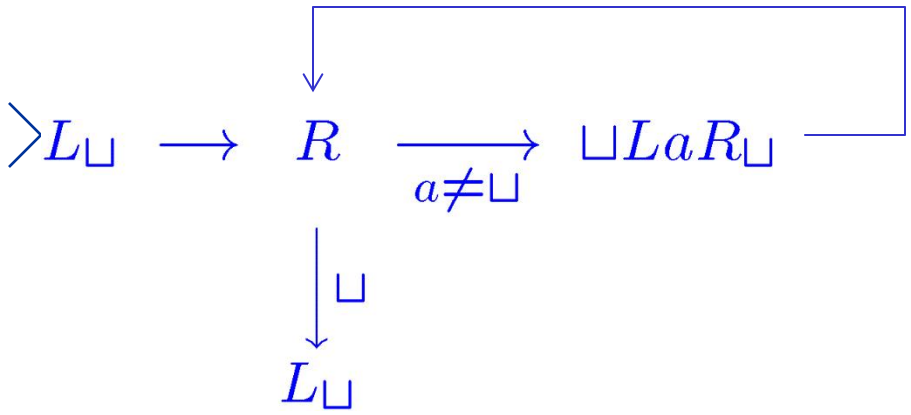
Example: (The Copying Machine)

Transforms $\sqcup w \sqcup$ into $\sqcup w \sqcup w \sqcup$ (w contains no blanks).

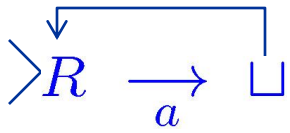


Example: (The left-shifting machine $S_{\leftarrow}$)

Transforms $\sqcup w \sqcup$ into $w \sqcup$ (w contains no blanks).



Example: Erases the $a's$ in tape.



4.2 Computing with Turing Machines

How TMs are Used (Deciding Languages)

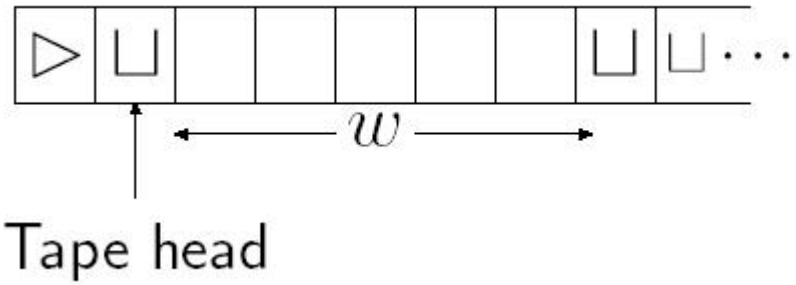
Question:

As language recognizers or decision procedures

i.e., to answer questions of the form: is $w \in L$?

Convention:

If $M = (K, \sum, \delta, s, H)$ is a TM and $w \in (\sum - \{\sqcup, \triangleright\})^*$, then the **initial configuration** of M on input w is $(s, \triangleright \underline{\sqcup} w)$.



(w *must not contain blanks*)

Definition: Let $M = (K, \sum, \delta, s, H)$ is a TM.

- $H = \{y, n\}$ consists of two distinguished halting states(y — "yes" and n — "no").

- Any halting configuration whose state component is y — *accepting configuration*;

- Any halting configuration whose state component is n — *rejecting configuration*.

Definition: (continued)

- M **accepts** $w \in (\sum - \{\sqcup, \triangleright\})^*$ if $(s, \triangleright \underline{\sqcup} w)$ yields an accepting configuration; M **rejects** w if $(s, \triangleright \underline{\sqcup} w)$ yields an rejecting configuration.
- Let $\sum_0 \subseteq (\sum - \{\sqcup, \triangleright\}$ be a alphabet — input alphabet of M.

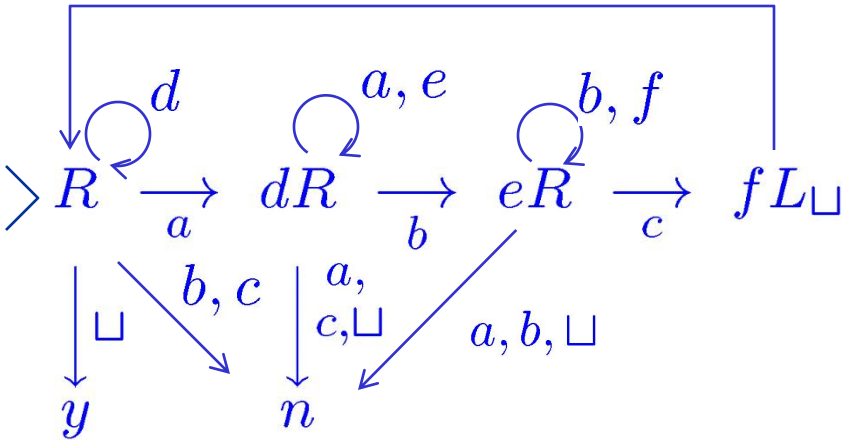
M **decides** $L \subseteq \sum^*$ if $\forall w \in \sum^*$ the following is true:

- $w \in L$ iff M accepts w;
- $w \notin L$ iff M rejects w.

- A language L is **recursive** if $\exists$ a TM that decides L.

Example:

Consider the language $L = \{a^n b^n c^n : n \geq 0\}$. The Turing Machine shown below decides L.



On input $a^n b^n c^n$, M will operates in n stages. In each stage M start from the left end and move to the right in search an a. When it finds an a, it replaces it by d and then looks further to the right for a b. When a b is found, it is replaced by a e, and then M looks for a c. When a c is found and it us replaced by a f, then the stage is over.

Remark:

- With FA and PDA, one of two things could happen: either the machine accepts the input, or rejects it.
- Turing Machine, even if it has only two halt states y and n, always has the option of evading an answer by failing to halt.

Remark:

- The word "recursive" is used because these are the same sets as can be defined via certain systems of recursive (self-referential) functions.

[Later $\cdots$]

- How many recursive languages are there?

How TMs are Used (Computing Functions)

TMs are often used to compute functions.

Definition: Let $M = (K, \sum, \delta, s, H)$ is a TM. Let $\sum_0 \subseteq \sum - \{\triangleright, \sqcup\}$ be an alphabet and $w \in \sum_0^*$.

- Suppose that M halts on input w and that $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (h, \triangleright \underline{\sqcup} y)$ for some $y \in \sum_0^*$. Then y is called the output of M on input w, denoted by $M(w)$.

Definition:(continued)

- Let f be a function f: $\sum_0^* \to \sum_0^*$. M **computes** function f if,

$$\forall w \in \sum_0^*,\ M(w) = f(w).$$

- A function f is **recursive** if $\exists$ a TM M computes f.

Example: The function $\kappa : \sum^* \to \sum^*$ defined as $\kappa(w) = ww$ can be computed by $CS_{\leftarrow}$.

The Class of Recursive Functions

A function from numbers to numbers is **recursive** if the string function on their binary notations is recursive.

Binary Notations:

- Strings in $\{0,1\}^*$ can be used to represent the nonnegative integers. $\forall w = a_1 a_2 \cdots a_n \in \{0,1\}^*$ represents the number:

$$num(w) = a_1 \cdot 2^{n-1} + a_2 \cdot 2^{n-2} + \cdots + a_n.$$

- Any natural number can be represented in a unique way by a string $0 \cup 1(0 \cup 1)^*$.

Definition: Let $M = (K, \sum, \delta, s, \{h\})$ is a TM.

- $0, 1, ; \in \sum$ and let $f : \mathbb{N}^k \rightarrow \mathbb{N}$ be any function for some $k \geq 1$. M **computes** f if, $\forall w_1, \cdots, w_k \in 0 \cup 1\{0,1\}^*$,
  $num(M(w_1; \cdots; w_k)) = f(num(w_1), \cdots, num(w_k))$.
- $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is **recursive** if $\exists$ a TM M computes f.

Remark:

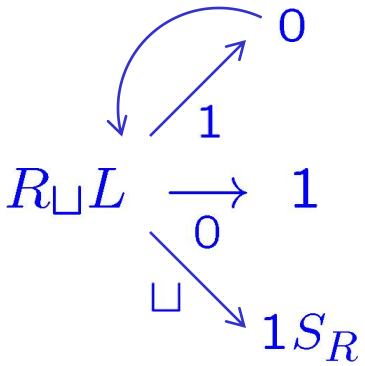
– M started with the binary representations of integers $n_1, n_2, \cdots, n_k$ as input.

– M eventually halts.

– When it does halt, its tape contains a string that represents number $f(n_1, n_2, \cdots, n_k)$.

Example: Design a machine to compute the successor function:

$$succ(n) = n + 1.$$



TM and Recursive Enumerable Languages

Definition: Let $M = (K, \sum, \delta, s, H)$ is a TM. Let $\sum_0 \subseteq \sum - \{\triangleright, \sqcup\}$ be an alphabet and $L \subseteq \sum_0^*$.

- M **semidecides** L if for $\forall w \in \sum^*$ the following is true:

$$w \in L \Leftrightarrow M \text{ halts on input } w.$$

- A language L is **recursively enumerable(r.e.)** iff $\exists$ a TM M that semidecides L.

Remark:

- L be a recursively enumerable:

  $w \in L \Leftrightarrow M$ halts.

  $w \notin L \Leftrightarrow M$ never enter the halting state.

  - *Recursion is related to recursion in programs*
  - *Enumerable means there is some TM that can enumerate the strings in the language.([Later $\cdots$])*

- Extending the functional notation:

  $M(w) = \uparrow$ if M fails to halt in input w.

Turing Machine semidecides L:

$$M(w) = \uparrow \Leftrightarrow w \notin L.$$

Example:

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

Then L is semidecides by the TM shown below.

Thus L is recursively enumerable.

Remark:

Design a Turing Machine that fail to halt by

- going on forever into the blanks
- looping forever

Some Properties

Theorem: The complement of a recursive language is recursive.

- Given M that halts on all inputs, construct M' that simulates M.
  - *If M halts and accepts, M' halts and does not accept*
  - *If M halts and does not accept, M halts and accepts*
- Let $M = (K, \sum, \delta, s, \{y, n\})$ and L is decided by M.

$\Rightarrow M' = (K, \sum, \delta', s, \{y, n\})$ decides $\sum^* - L$.

$$\delta'(q,a) = \begin{cases} n & \text{if } \delta(q,a) = y \\ y & \text{if } \delta(q,a) = n \\ \delta(q,a) & \text{otherwise.} \end{cases}$$

Theorem: If L is a recursive language, then it is recursively enumerable.

Let $M = (K, \sum, \delta, s, \{y, n\})$ and L is decided by M.
$\Rightarrow$ Construct $M' = (K, \sum, \delta', s, \{y\})$ semidecides L.

$$\delta'(q, a) = \begin{cases} n & \text{if } q = n \\ \delta(q, a) & \text{otherwise.} \end{cases}$$

Theorem: The union/intersection of two recursive languages is recursive.

4.3 Extension of Turing Machines

Extension of Turing Machines:

- Multiple tapes
- Two-way infinite tape
- Multiple heads
- Multi-dimensional tape
- Non-determinism

– These "enhancements" do not increase the power of TMs.
– Using the additional features, we can easily design Turing Machines to solve particular problems.

Multiple tapes

Definition: Let $k \geq 1$ be an integer. A k-**tape TM** is a quintuple $(K, \sum, \delta, s, H)$, where $K, \sum, s$ and H are as in the definition of the ordinary TM, and δ, the transition function:

$$\delta : (K - H) \times \sum^k \rightarrow K \times (\sum \cup \{\leftarrow, \rightarrow\})^k.$$

Definition: Let $M = (K, \sum, \delta, s, H)$ be a k-tape TM. A **configuration** of M is a member of:

$$K \times (\triangleright \sum^* \times (\sum^* (\sum - \{\sqcup\}) \cup \{e\}))^k.$$

Convention:

– The input string is placed on the first tape;
– The other tapes are iniatially blank, with the head on the leftmost blank square of each;
– At the end of the computation, the output on the first tape; the other tapes are ignored.

Example: (The Copying Machine)

Transforms $\sqcup w \sqcup$ into $\sqcup w \sqcup w \sqcup$ (w contains no blanks).

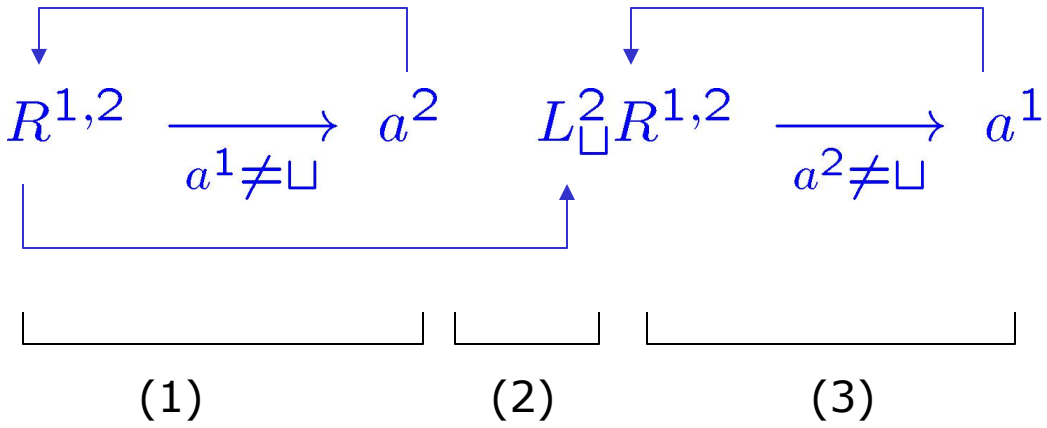
Solution:

2-tape Turing machine can accomplish this as follows.

1) Move the heads on the both tapes to the right, copy each symbol on the first tape onto the second tape, until a blank is found on the first tape.

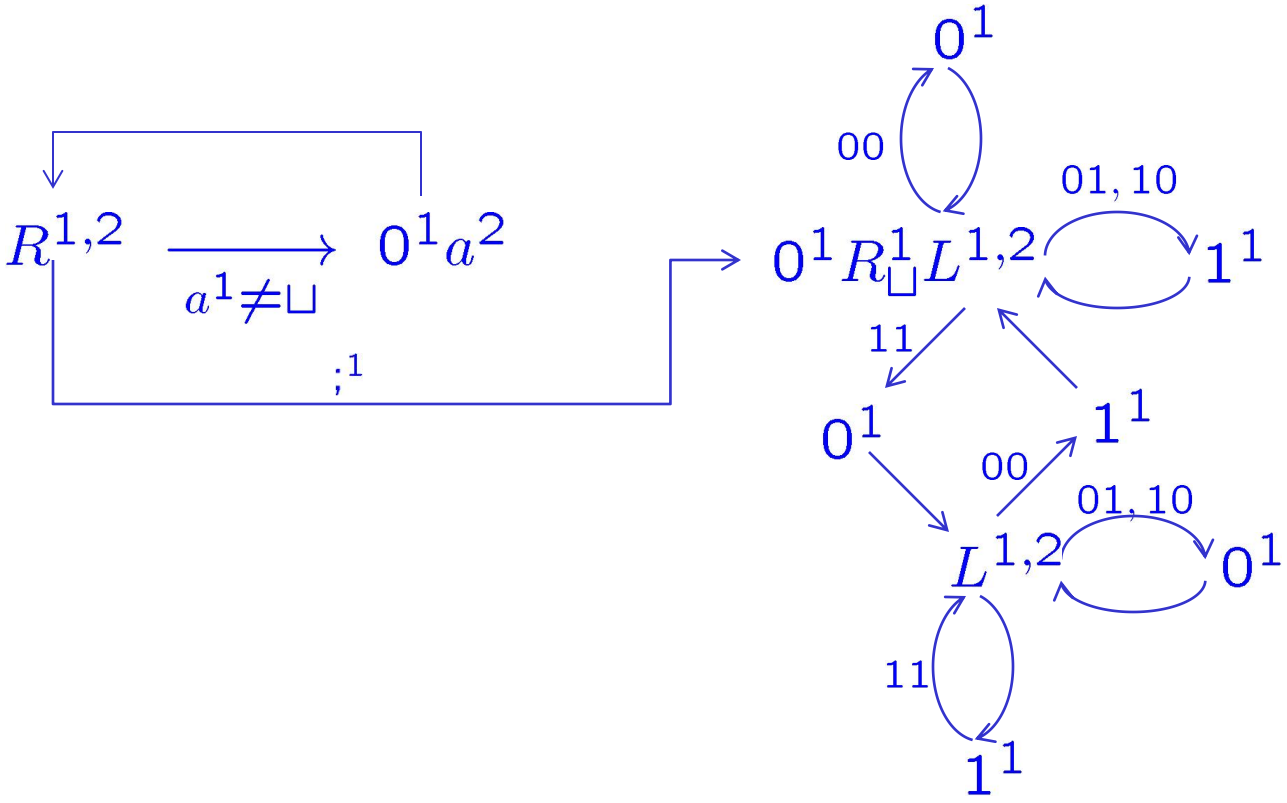
2) Move the heads on the second tape to the left until a blank is found.

3) Again move the heads on the both tapes to the right, copy symbols from the second tape onto the first tape. Halt when a blank is found on the second tape.



	At beginning	After (1)	After (2)	After (3)
First tape	$\triangleright \underline{\sqcup} w$	$\triangleright \sqcup w \underline{\sqcup}$	$\triangleright \sqcup w \underline{\sqcup}$	$\triangleright \sqcup w \sqcup w \underline{\sqcup}$
Second tape	$\triangleright \underline{\sqcup}$	$\triangleright \sqcup w \underline{\sqcup}$	$\triangleright \underline{\sqcup} w$	$\triangleright \sqcup w \underline{\sqcup}$

Example: (Add arbitrary any binary numbers)



k-tape TM/Standard TM Equivalence

Theorem: Let $M = (K, \sum, \delta, s, H)$ be a k-tape TM($k \geq 1$). Then there is a standard TM $M = (K', \sum', \delta', s', H)$, where $\sum \subseteq \sum'$ and such that the following holds:

1) For any input string $x \in \sum^*$, M on input x halt with output y on the first tape iff M' on input x halts as the same halting state, and with the same output y on its tape.

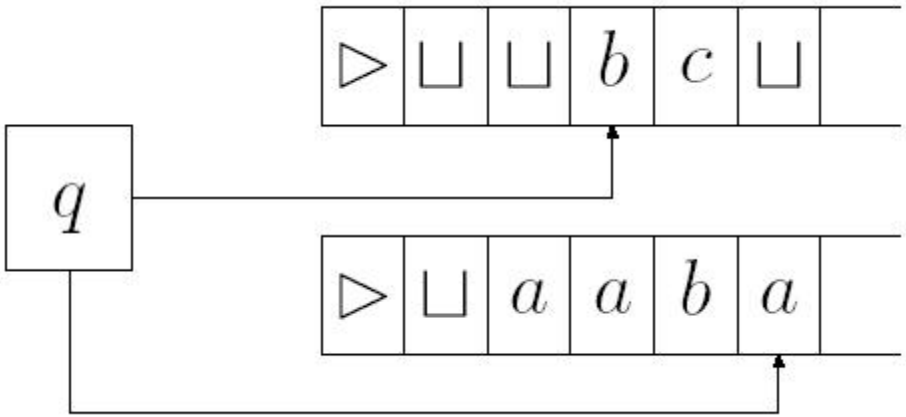
2) If M halts on input x after t steps, then M' halts on input x after a number of steps which is $\mathscr{O}(t \cdot (|x| + t))$.

Proof:

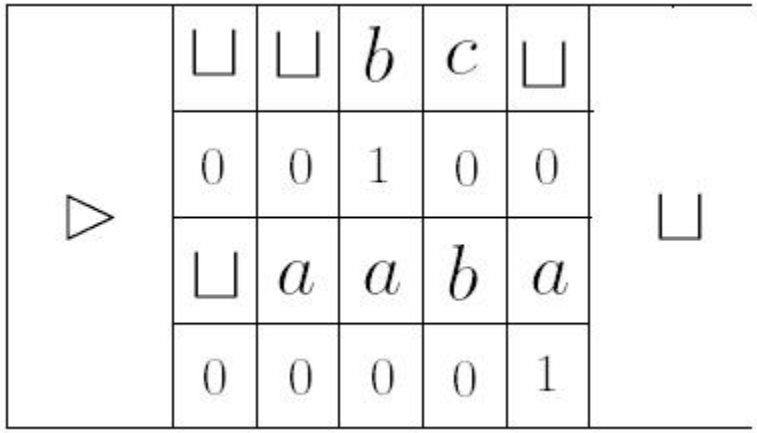
Main idea:

Simulate a k-tape TM by a one-tape TM whose tape is split (conceptually) into $2k$ tracks:

- k tracks for tape symbols
- k tracks for head positions
  (one in each track)



2-tape Turing Machine



If the head on the ith tape of M is positioned over the nth tape square, then the $2i$th track of tape M' contains a 1 in the $(n+1)$st tape square and a 0 in all tape squares except $(n+1)$st.

Let $M = (K, \sum, \delta, s, H)$ be a k-tape TM

$\Rightarrow$ Construct a standard TM $M' = (K', \sum', \delta', s', H)$

- $\sum' = \sum \cup (\sum \times \{0, 1\})^k$
  - *the alphabet of M' consists of alphabet of M, and $2k$-tuples $(a_1, b_1, \ldots, a_k, b_k)$, with $a_1, \ldots, a_k \in \sum$, $b_1, \ldots, b_k \in \{0, 1\}$.*

- When given a input $w \in \sum^*$, M' operates as follows:

I. Representation of the initial configuration of M:

– Shift the input one step square to the right.

– Return to the square immediately to the right of the $\triangleright$, and write the symbol $(\sqcup, 1, \ldots, \sqcup, 1)$ — this signifies that the first squares of all k tapes contain a $\sqcup$ and are all scanned by the heads.

– Proceed to the right, at each square, if $a \neq \sqcup$ is encountered, write $(a, 0, \sqcup, 0, \ldots, \sqcup, 0)$ in its position. If $a = \sqcup$ is encountered, the first phase is over.

II. Simulate the computation of M:

(until M would halt, if it would halt)

To simulate one step of M, M' will perform:

It starts each step simulation with its head scanning the first square of its tape that has not yet been subdivided into tracks

(a) **Scan left down the tape**, gather information about the symbols have scanned by the k tapes heads of M. After all scanned symbols have been identified, **return to the leftmost true blank** and then **change state** of the finite control to reflect the k-tuple of symbols form $\sum$ in k tracks at the marked head positions.

(b) Scan left then rightdown the tape to update the tracks in accordance with the move of M.

III. Representation of halted configuration

When M would halt:

– M' first converts its tape from tracks into single symbol format, ignoring all tracks except for the first.

– M' positions its head where M would have placed its first head.

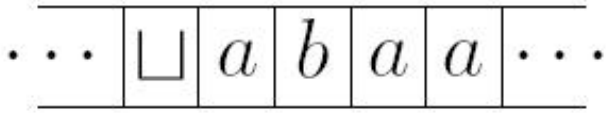
– M' halts in the same state as M would have halted.

- Time Complexity: $\mathscr{O}(t(|x|+t))$

Simulating t steps of M on input x, M':
Phase 1: $\mathscr{O}(|x|)$
Phase 2: For each step of M, M' $\mathscr{O}(|x|+2+t)$.

Corollary: Any function that is computed or language that is decided or semidecided by a k-tape TM is also computed, decided, or semidecided, respectively, by a standard TM.

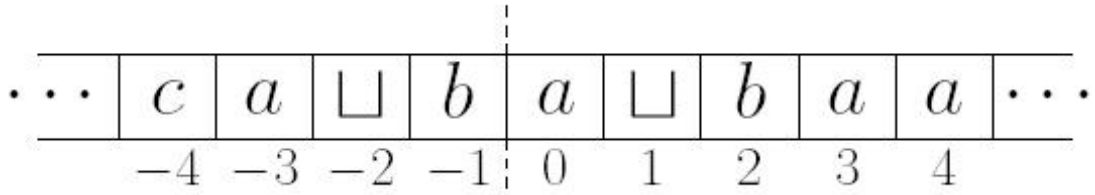
Two-way infinite tape

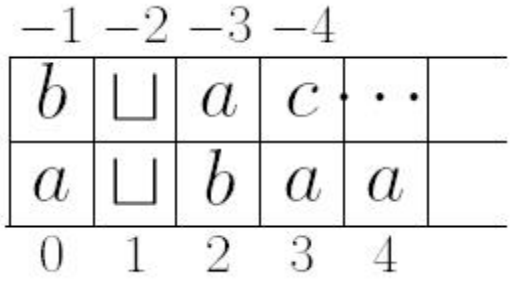


- Unbounded tape to left as well as right
- Same convention for input/output

• TMs with 2-way infinite tape are no more powerful than standard TM.

- A 2-way infinite tape can be easily simulated by a two-tape machine.





Multiple Heads

- A multiple TM: one tape and several heads.
- In one step, the heads all sense the scanned symbols and move or write independently.
- A multiple heads TM can be easily simulated by a k-tape machine.

Basic idea:

– divide the tape into tracks, all but one of which are used solely to record the head positions.

– To simulate one step by multiple heads TM, the tape must be scanned twice: once to find the symbols at the head positions, and again to change those symbols or move the heads as appropriate.

Two Dimensional Tape

- Allow its tape to be an infinite two-dimensional grid.

- much more useful than standard TM to solve problems such as "zigsaw puzzles".

Theorem: Any language decided or semidecided and any function computed by TM with several tapes, heads, two-way infinite tapes or multi-dimensional tapes, can be decided, semidecided, or computed respectively, by a standard TM.

4.4 Nondeterministic Turing Machines

Definition: A **nondeterministic Turing Machine(NTM)** is a quintuple $M = (K, \sum, \Delta, s, H)$, where $K, \sum$ and H are as for standard TMs, and Δ is a subset of

$$((K - H) \times \sum) \times (K \times (\sum \cup \{\leftarrow, \rightarrow\})).$$

- Like TMs, but is now a **relation** rather than a **function**.

- Configurations and the relations $\vdash_M$ and $\vdash_M^*$ are defined in the similar way in standard TM.
  - $\vdash_M$ *need not be single-valued.*

□ NTM Decides Languages/Computes Functions

Definition: Let $M = (K, \sum, \Delta, s, H)$ is a NTM.

- M **accepts** an input $w \in (\sum - \{\triangleright, \sqcup\})^*$ if $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (h, u\underline{a}w)$ for some $h \in H$ and $a \in \sum, u, w \in \sum^*$.
- M **semidecides a language** $L \subseteq (\sum - \{\triangleright, \sqcup\})^*$ if the following holds for all $w \in (\sum - \{\triangleright, \sqcup\})^*$:

$$w \in L \text{ if and only if } M \text{ accepts } w.$$

Remark:

NTM M accepts an input w:

– as long as at least one halting computation exists.

Definition: Let $M = (K, \sum, \Delta, s, \{y, n\})$ be a NTM.

• M **decides a language** $L \subseteq (\sum - \{\triangleright, \sqcup\})^*$ if the following two conditions hold for all $w \in (\sum - \{\triangleright, \sqcup\})^*$:

(a) $\exists N(M, w) \in \mathbb{N}$ such that there is no configuration C satisfying $(s, \triangleright \underline{\sqcup} w) \vdash_M^N C$.

(b) $w \in L$ iff $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (y, u\underline{a}v)$ for $a \in \sum$, and $u, v \in \sum^*$.

• M **computes a function** $f : (\sum - \{\triangleright, \sqcup\})^* \rightarrow (\sum - \{\triangleright, \sqcup\})^*$ if the following two conditions hold for all $w \in (\sum - \{\triangleright, \sqcup\})^*$:

(a) $\exists N(M, w) \in \mathbb{N}$ such that there is no configuration C satisfying $(s, \triangleright \underline{\sqcup} w) \vdash_M^N C$.

(b) $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (h, u\underline{a}v)$ iff $ua = \triangleright\sqcup$ and $v = f(w)$.

Remark:

- For NTM to decide a language and compute a function, all of its computations halt(condition(a)).

- For NTM decide a language, we only require that at least one of its possible computations end up accepting the input.

- For NTM to compute a function, we require that all possible computations agree on the outcome.

Example:

Design a NTM to semidecide $C = \{num(p \cdot q) : p, q \geq 2\}$.

– set of composite numbers

Solution:

• Design a NTM to semidecide C by guessing the factors, if there are any.

• The machine operates as follows:

(1) Nondeterministically choose two binary numbers p, q (except 0,1), and write their binary representation next to the input.

(2) Use the multiplication machine to replace the binary representation of p and q by $p \cdot q$.

(3) Check to see the two integers, n and $p \cdot q$, are same. Halt if the two integers are equal; otherwise continue forever.

Remark:

This machine can be modified to NTM that **decides** the language C.

NTM/Standard TM equivalent

Theorem: If a NTM M semidecides or decides a language, or computes a function, then there is a standard TM M' semidecides or decides the same language, or computes the same function.

Proof:

- We shall describe the construction for the case in which M semidecides a language L.
- Given a NTM M, we must construct a standard TM M' that determines, on input w, whether M has a halting computation on input w.

M may have an infinty of different computations starting from the same input; how can M explore them all?

- M' systemtically tries
  - all one-step computations
  - all two-step computations
  - all three-step computations

    ⋮ — *dovetailing procedure*

- Ultimately M' either:
  - discovers a halting computation of M, and halts itself,

or

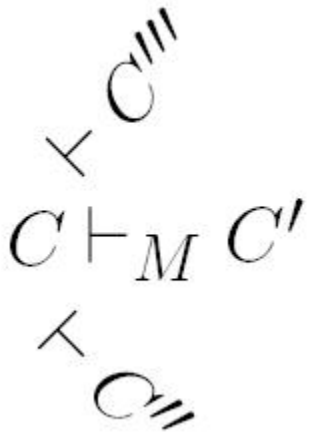
  - searches forever, and does not halt.

- There is a bounded number of k-step computations, for each k.

(*for each configuration there are only a constant number of "next" configurations in one step*)

The number of quadruples $(q, a, p, b) \in \Delta$ that can be applicable at any point is finite

— it cannot exceed $|K| \cdot (|\sum| + 2)$



M' in detail

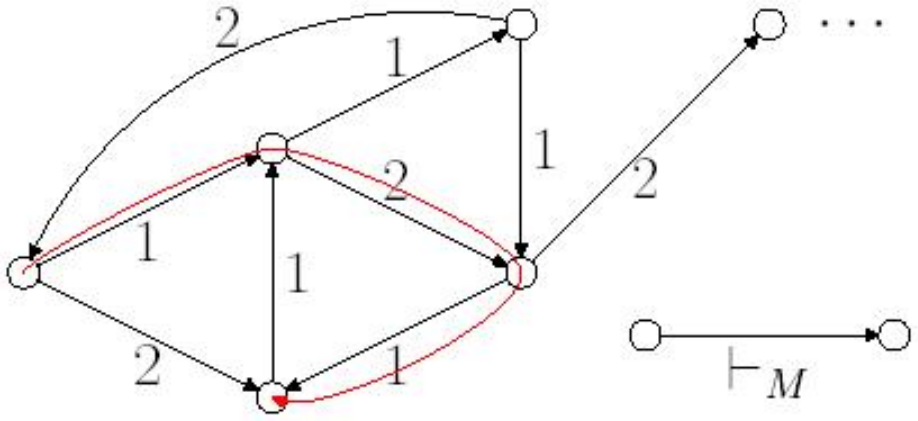
- Suppose that the maximum number of different transitions for a given state-symbol combination (q, a) is r.

- Number those transitions $1, \cdots, r$ (or less)

$$\begin{array}{ll} ((q,a),(p_1,b_1)) & (1) \\ ((q,a),(p_2,b_2)) & (2) \\ \vdots & \vdots \\ ((q,a),(p_r,b_r)) & (r) \end{array}$$

- Any computation of k steps is determined by a sequence of k numbers $\leq r$.

Configuration Space

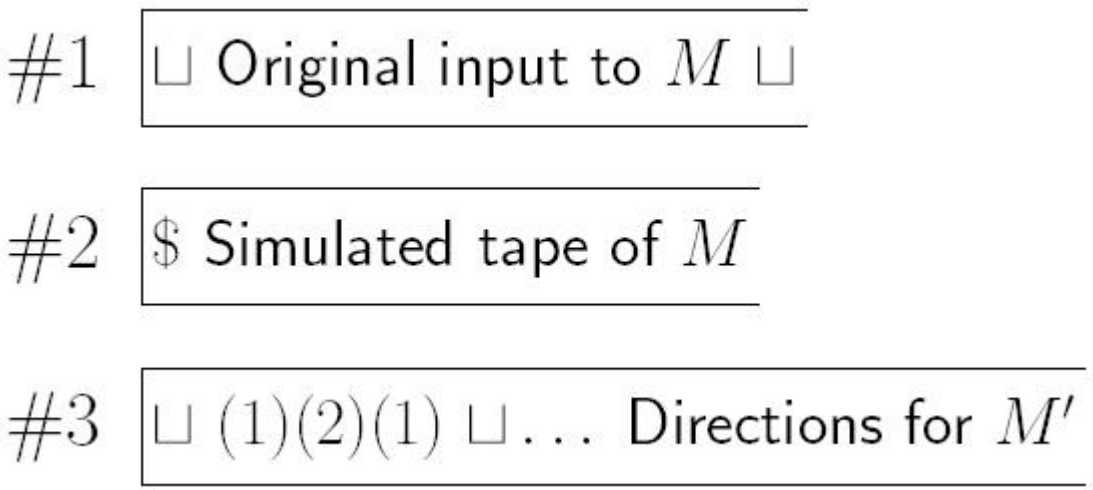


$\longrightarrow = (1)(2)(1)$

- Define the deterministic version of M: M_d
  - M_d has the same state of M
  - M_d has two tapes:
    - *The first tape is the tape of M, initially containing the input w.*
    - *The second tape initially contains a string of n integers in range $1, \cdots, r$, say $i_1 i_2 \cdots i_n$.*
  - M_d operates as folllows for n steps:
    - *In the first step, M_d chooses the i_1th — (p_{i_1}, b_{i_1}) among the r next state-action combinations and then applies to the initial configuration. Moves its second tape head to the right, so that it next scans i_2*
    - *In the next step, M_d takes the i_2th combination, then i_3th and so on.*
    - *When M_d sees a blank on its second tape, it halts.*

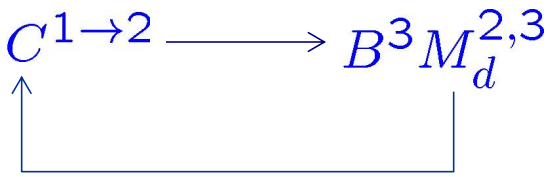
- How M' works: 3 tapes

(M' can be converted into an equivalent standard Turing Machine)



- The three tapes of M' are used as follows:
  - The first always contains the original input w.
    — never changed.
  - The second and the third tapes are used to simulate the computations of M_d with all strings in $\{1, \cdots, r\}^*$.
    – *The input is copied from the first tape onto the second before M' begins to simulate a new computation.*
    – *Initially, the third tape contain e.*
  - Between two simulations of M_d, M' uses a TM N to generate the lexicographically next string in $\{1, \cdots, r\}^*$.
    – *N will generate from e the strings $1, \cdots, r$, $11, 12, \cdots rr$, $111, \cdots$*

- M' is the Turing Machine given in the following Figure:



– $C^{1\to 2}$ *:a simple Turing Machine that erases the second tape and copies the first tape on the second.*
– B^3*: is the machine that generates the lexicographically next string in the third tape.*
– $M_d^{2,3}$*: is the deterministic version of* M*, operating on tape 2 and 3.*

- M' halt on input w iff (some computation of) M halts.

  – Suppose that M' indeed halts on input w

  $\Rightarrow$ *M_d halts and the third tape not scanning a blank*

  $\Rightarrow$ *For some string $i_1 \cdots i_n \in \{1, \cdots r\}^*$, M_d when started with w on its first tape and $i_1 \cdots i_n$ on its second, halts before reaching the blank*

  $\Rightarrow$ *There is a computation of M on input w that halts.*

  – There is a halting computation of M on input w with n steps

  $\Rightarrow$ *M' after at most $r + r^2 + \cdots + r^n$ failed attempts, the string in $\{1, \cdots, r\}^*$ corresponding to the choices of M's halting computation will be generated by B^3, and M_d will halt scanning the last symbol of this string.*

Remark:

- The simulation of a NTM by a deterministic one is not a step by step simulation, as well all other simulation we have seen.

- It requires **exponentially many steps** in n to simulation of n step by the NTM — whereas all other simulation described in this chapter are in **polynomial**.

4.5 Grammars

- Consider **language generator**:

  regular expression, context-free grammar, $\cdots$

- Like context-free grammars, except that if $u \rightarrow v$ is a rule, then u may be any string containing a nonterminal.

- So the rule $AXY \rightarrow AYX$ where $A, X, Y \in V - \sum$, "means" that the two-symbol substring XY can be replaced by YX whenever it appears with an A to its left.

Definition: A **grammar** (or **unrestricted grammar**) is a quadruple $G = (V, \sum, R, S)$,where

V is an alphabet;

$\sum \subseteq V$ is the set of **terminal** symbols, and $V - \sum$ is called the set of **nonterminal** symbols;

$S \in V - \sum$ is the **start** symbol; and

R is the set of **rules**, a finite subset of $(V^*(V - \sum)V^*) \times V^*$.

- $u \rightarrow_G v \Leftrightarrow (u, v) \in R.$
- $u \Rightarrow_G v \Leftrightarrow \exists w_1, w_2 \in V^*$, and some rule $u' \rightarrow_G v'$ such that $u = w_1 u' w_2, v = w_1 v' w_2.$

- $\Rightarrow_G^*$ is the reflexive, transitive closure of $\Rightarrow_G$.

- The **language generated by** G,
$$L(G) = \{w \in \sum^* : S \Rightarrow_G^* w\}.$$

- $w_0 \Rightarrow_G w_1 \Rightarrow_G \cdots \Rightarrow_G w_n$
— a **derivation** in G of w_n from w_0. n be the length of the derivation.

Remark:

Any context-free grammar is a grammar.

Example:

The grammar G generates the language $\{a^n b^n c^n : n \geq 1\}$.

Solution:

- $\sum = \{a, b, c\}$
- $V = \{a, b, c, S, A, B, C, T_a, T_b, T_c\}$
- $R = \{S \to ABCS, S \to T_c$

  (Thus $S \Rightarrow (ABC)^n T_c$, for any $n \geq 0$).

  $BA \to AB, CA \to AC, CB \to BC$

  (Any inversions of the proper order can be repaired)

  $CT_c \to T_c c, CT_c \to T_b c$

  (The c-transformer can crawl to the left, and turn into a b-transformer)

  $BT_b \to T_b b, BT_b \to T_a b, AT_a \to T_a a, T_a \to e\}$

- The only way to get a string of terminals yields one of the form $a^n b^n c^n$.

□ Grammar/Turing Machine equivalent

Theorem: A language is generated by a grammar if and only if it is recursively enumerable.

Proof:

($\Rightarrow$) L is generated by a grammar $\Rightarrow$ L is r.e..

Let $L = L(G)$, G a grammar. To construct an NDTM M such that $L(M) = L(G)$.

- Let M have three tapes.
  - The first tape permanently holds input w
  - In the second tape M tries to reconstruct a derivation of w from S in grammar G; M therefore starts by writing S on the second tape.

- $|R| + 1$ possible states:
  - Each of the first $|R|$ states is the beginning of a sequence of transitions that applies the correponding rule to the current contents of the second tape.
  - The $|R|+1$st choice of M: Check whether the current string equals w. If so, M halts and accpets; if the strings are found unequal, M again loops forever.

- M nondeterministically carries out a derivation $S = w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \cdots$, checking each step to see if $w_i = w$.

($\Leftarrow$) L is r.e. $\Rightarrow$ L is generated by a grammar G.

Let $M = (K, \sum, \delta, s, \{h\})$ is a Turing machine.
To construct Grammar G such that G generates the language $L \subseteq (\sum - \{\sqcup, \triangleright\})^*$ semidecided by M.

Convention:

- K and $\sum$ are disjoint, and neither contains the new endmarker symbol $\triangleleft$.
- If M halts, it always halts in the configuration $(h, \triangleright\underline{\sqcup})$.

Main idea:

G will simulate a **backwards computation** by M. The intermediate strings will be coded forms of configurations:

Configuration	String
$(q, \triangleright u\underline{a}w)$	$\triangleright uaqw\triangleleft$

Building a Grammar from a TM

$G = (V, \sum - \{\sqcup, \triangleright\}, R, S)$, where

- $V = \sum \cup K \cup \{S, \triangleleft\}$.
- R contains the following rules:
  - If $(q, a) = (p, b)$, where $b \in \sum$, then G has a rule: $bp \rightarrow aq$.
  - If $(q, a) = (p, \rightarrow)$, then G has
    $abp \rightarrow aqb$, for each $b \in \sum$
    $a \sqcup p\triangleleft \rightarrow aq\triangleleft$ (moving right onto blank tape)

- If $(q, a) = (p, \leftarrow)$, then G has
  $pab \rightarrow aqb$, if $a \neq \sqcup$
  $p \sqcup b \rightarrow \sqcup qb$ and $p\triangleleft \rightarrow \sqcup q\triangleleft$, if $a = \sqcup$

- Since $(h, \sqcup)$ is the final configuration,
  $S \rightarrow \triangleright \sqcup h\triangleleft$

- Finally, if s is the start state of the TM,
  $\triangleright \sqcup s \rightarrow e$
  $\triangleleft \rightarrow e$

Claim:

For any two configurations $(q_1, u_1\underline{a_1}w_1)$ and $(q_2, u_2\underline{a_2}w_2)$ of M, we have that $(q_1, u_1\underline{a_1}w_1) \vdash_M (q_2, u_2\underline{a_2}w_2)$ iff $u_2a_2q_2w_2\triangleleft \Rightarrow_G u_1a_1q_1w_1\triangleleft$

We now complete the proof of the theorem, by showing that, for all $w \in (\sum - \{\triangleright, \sqcup\})^*$, M halts on w iff $w \in L(G)$.

$$
\begin{aligned}
w \in L(G) \text{ iff } & S \Rightarrow_G \triangleright \sqcup h\triangleleft \Rightarrow_G^* \triangleright \sqcup sw\triangleleft \Rightarrow_G w\triangleleft \Rightarrow_G w \\
\text{iff } & (s, \triangleright\underline{\sqcup}w) \vdash_M^* (h, \triangleright\underline{\sqcup}) \\
\text{iff } & w \in L(M)
\end{aligned}
$$

Definition: Let $G = (V, \sum, R, S)$ be a grammar, and let $f : \sum^* \to \sum^*$ be a function.

- G **computes** f, if for all $w, v \in \sum^*$, the following is true:

$$SwS \Rightarrow_G^* v \text{ iff } v = f(w).$$

- A function $f : \sum^* \to \sum^*$ is called **grammatically computable** iff there is a grammar G that computes it.

Theorem: A function $f : \sum^* \to \sum^*$ is **recursive** iff and only if it is grammatically computable.

4.6 Numerical Functions

- The primitive recursive functions were first formally defined by Gödel in 1929.

- Like FA, they are a simplified model of computation.

- There are some 'primitive' functions that we definitly want to consider computable:

  the projection functions
  the zero function
  the successor function

  — **initial functions or basic functions.**

  — **Two operations:composition and recurisive.**

Definition:

- The **basic functions** are the following :

  (a) For any $k \geq 0$, the k**ary zero function** is defined as

$$zero_k(n_1, \ldots, n_k) = 0$$

for all $n_1, \ldots, n_k \in \mathbb{N}$.

  (b) For $k \geq j > 0$, the j**th** k**-ary identity function** is simply the function

$$id_{k,j}(n_1, \ldots, n_k) = n_j$$

for all $n_1, \ldots, n_k \in \mathbb{N}$.

  (c) The **successor function** is defined as

$$succ(n) = n + 1$$

for all $n \in \mathbb{N}$.

Definition

• Two simple ways of combining functions to get slightly more complex functions.

(1) For $k, l \geq 0$, let $g : \mathbb{N}^k \to \mathbb{N}$ be a k-ary function, let $h_1, \ldots, h_k$ be l-ary functions. Then the **composition of** g **with** $h_1, \ldots, h_k$ is the l-ary function defined as

$$f(n_1, \ldots, n_l) = g(h_1(n_1, \ldots, n_l), \ldots, h_k(n_1, \ldots, n_l)).$$

(2) For $k \geq 0$, let g be a k-ary function, and let h be a (k+2)-ary function. Then the **function defined by** g **and** h is the $k+1$-ary function f defined as

$$f(n_1, \ldots, n_k, 0) = g(n_1, \ldots, n_k)$$

$$f(n_1, \ldots, n_k, m+1) = h(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m))$$

for all $n_1, \ldots, n_k, m \in \mathbb{N}$.

Definition: (Continued)

• The **primitive recursive functions** are all basic functions, and all functions that can be obtained by them by any number of successive applications of composition and recursive definition.

Example: The functions $plus2(n) = n + 2$ is primitive recursive.

$$plus2(n) = succ(succ(id_{1,1}(n))).$$

Example: The following functions are primitive recursive.

(1) $plus(m, n) = m + n$

(2) $mult(m, n) = m \cdot n$

(3) $exp(m, n) = m^n$

Solution:

(1)
$$plus(m, 0) = m$$
$$plus(m, n + 1) = succ(plus(m, n))$$

(2)
$$mult(m, 0) = zero(m)$$
$$mult(m, n + 1) = plus(m, mult(m, n))$$

(3)
$$exp(m, 0) = succ(zero(m))$$
$$exp(m, n + 1) = mult(m, exp(m, n)).$$

Example: All constant functions and the sign function are primitive recursive.

Solution:

- $f(n_1, \ldots, n_k) = m$ can be obtained by

$$f(n_1, \ldots, n_k) = succ(\ldots succ(zero(n_1, \ldots, n_k)) \ldots)$$

- $sgn(0) = 0$, and $sgn(n + 1) = 1$.

Example: Function

$$m \sim n = \max\{m - n, 0\}$$

is primitive recursive.

Solution: First define the predecessor function:

$$pred(0) = 0$$

$$pred(n + 1) = n$$

Then we have

$$m \sim 0 = m$$

$$m \sim (n + 1) = pred(m \sim n)$$

Definition: A **primitive recursive predicate** be a primitive recursive function that only takes values 0 and 1.

Example: The following predicats are primitive recursive.

- $iszero(n)$: which is 1 if $n = 0$, and 0 if $n > 0$.

$$(iszero(0) = 1,\ iszero(m+1) = 0)$$

- $positive(n) = sgn(n)$
- $greater\text{-}than\text{-}or\text{-}equal(m,n)(m \geq n) = iszero(n \sim m)$
- $less\text{-}than(m,n) = 1 \sim greater\text{-}than\text{-}or\text{-}equal(m,n)$.

Remark:

• The negation of any primitive recursive predicate is also a primitive recursive predicate.

$p(m)$ is a primitive recursive predicate
$\neg p(m) = 1 \sim p(m)$

• The **disjunction** and **conjunction** of two primitive recursive predicates are also primitive recursive.

$p(m,n)$, $q(m,n)$ are primitive recursive predicates
$p(m,n) \vee q(m,n) = 1 \sim iszero(p(m,n) + q(m,n))$
$p(m,n) \wedge q(m,n) = 1 \sim iszero(p(m,n) \cdot q(m,n))$

- If f and g are primitive recursive functions and p is a primitive recursive predicate, then the function **defined by cases**

$$f(n_1,\ldots,n_k)=\begin{cases} g(n_1,\ldots,n_k), & \text{if } p(n_1,\ldots,n_k);\\ h(n_1,\ldots,n_k), & \text{otherwise}\end{cases}$$

is also primitive recursive.

$f(n_1,\ldots,n_k)$ can be rewitten as:

$$f(n_1,\ldots,n_k)=p(n_1,\ldots,n_k)\cdot g(n_1,\ldots,n_k)+(1\sim p(n_1,\ldots,n_k))\cdot h(n_1,\ldots,n_k)$$

Example: We can define div and rem

$$rem(0, n) = 0$$

$$rem(m+1, n) = \begin{cases} 0, & \text{if } equal(rem(m, n), pred(n)); \\ rem(m, n) + 1, & \text{otherwise} \end{cases}$$

and

$$div(0, n) = 0$$

$$div(m+1, n) = \begin{cases} div(m, n) + 1, & \text{if } equal(rem(m, n), pred(n)); \\ div(m, n), & \text{otherwise} \end{cases}$$

Example: Function $digit(m, n, p)$,

— *the m-th least significant digit of the base-p representation of n*

is primitive recursive.

$$n = a_l p^l + \cdots + a_{m-1} p^{m-1} + \cdots + a_1 p + a_0$$

Since

$$digit(m, n, p) = div(rem(n, p \uparrow m), p \uparrow (m \sim 1))$$

Specially,

$$odd(n) = digit(1, n, 2)$$

Example: If $f(n,m)$ is primitive recursive function, then the sum

$$sum_f(n,m) = \sum_{k=0}^{m} f(n,k)$$

is also primitive recursive.

- Since it can be defined as

$$sum_f(n,0) = 0$$
$$sum_f(n,m+1) = sum_f(n,m) + f(n,m+1).$$

- Similarly,

$$mult_f(n,m) = \prod_{k=0}^{m} f(n,k)$$

Example: If $p(n_1, \ldots, n_k, m)$ is primitive recursive predicate, then

$$\exists t_{(\leq m)} p(n_1, \ldots, n_k, t) \text{ and } \forall t_{(\leq m)} p(n_1, \ldots, n_k, t)$$

are primitive recursive predicates.

$$\exists t_{(\leq m)} p(n_1, \ldots, n_k, t) \Leftrightarrow \sum_{t=0}^{m} p(n_1, \ldots, n_k, t) \neq 0$$

$$\forall t_{(\leq m)} p(n_1, \ldots, n_k, t) \Leftrightarrow \prod_{t=0}^{m} p(n_1, \ldots, n_k, t) \neq 0$$

— **bounded Conjunctions (Disjunctions) of Predicates**

Example:

Predicate $y \mid x$ (x is divided exactly by y) is primitive recursive predicates.

- $y \mid x \Leftrightarrow \exists t_{(\leq x)}(y \cdot t = x)$

Example:

Predicate $prime(x)$ (x is prime) is primitive recursive predicates.

- $prime(x) \Leftrightarrow (x > 1) \wedge \forall t_{(<x)}[t = 1 \vee \neg(t|x)]$

Example: The set of primitive recursive functions is enumerable.

- Each primitive recursive function can in principle be defined in terms of the basic functions, and therefore can be represented as a string in finite alphabet; the alphabet should contain symbols for the identity, successor, and zero functions, for primitive recursion and composition, plus parentheses and the symbols 0 and 1 used to index in binary basic functions.

- We could enumerate all strings in the alphabet and keep the ones that are legal definitions of primitive recursive functions.

- Suppose that we list all unary primitive recursive functions, as strings, in lexicographic order

$$f_0, f_1, f_2, f_3, \ldots$$

Define a function $g(n) = f_n(n) + 1$.

– For any number $n \geq 0$, find the n-th primitive recursive function in the list, f_n, and use its definition to compute the number $f_n(n)+1$.

$g(n)$ is computable

$g(n)$ is not primitive recursive

Because if it were, say $g = f_m$ for some $m \geq 0$ then we would have $f_m(m) = f_m(m) + 1$, which is absurd.

Remark: The set of primitive recursive function is proper subset of the set of recursive function.

Minimalization

Definition: Let g be a $(k+1)$-ary function, for some $k \geq 0$.

- The **minimalization** of g is the k-ary function f defined as follows:

$$f(n_1, \ldots, n_k) = \begin{cases} \text{the least m such that } g(n_1, \ldots, n_k, m) = 1 \\ \text{if such m exists;} \\ \text{0 otherwise} \end{cases}$$

We shall denote the minimalization of g by

$$\mu\ m[g(n_1, \ldots, n_k, m) = 1].$$

Definition: (Continued)

• Although the minimalization of a function g is always well-defined, there is no obvious method for computing it — even if we konw how to compute g. The obvious method

$m := 0$;
while $g(n_1, \ldots, n_k, m) \neq 1$ *do* $m := m + 1$;
output m

is **not** an algorithm, because it may fail to terminate.

• A function g is **minimalizable** if the above method always terminates. That is, a $(k+1)$-ary function g is minimalizable if it has the following properties:

For every $n_1, \ldots, n_k \in \mathbb{N}$, $\exists m \in \mathbb{N}$ such that $g(n_1, \ldots, n_k, m) = 1$.

Definition: (Continued)

- A function is μ-**recursive** if it can be obtained from the basic functions by the operations of composition, recursive definition, and minimalization of minimalizable functions.

Remark:

We cannot apply the diagonalization principle to show that there is a computable function that is not μ-recursive.

— It is not clear at all whether all applications of minimalization in the definition indeed acted upon minimalizable functions!

Example: Using minimalization, we can define the $logarithm$ function:

$$\log(m,n) = \mu\ p[greater\text{-}than\text{-}or\text{-}equal((m+2)\uparrow p, n+1)].$$

- $\log(m,n) = \lceil \log_{(m+2)}(n+1) \rceil$

— *to avoid the mathematical pitfalls in the definition of* $\log_m n$*,when* $m \leq 1$ *or* $n = 0$.

- This is a proper definition of a μ-recursive function, since the function $g(m,n,p) = greater\text{-}than\text{-}or\text{-}equal((m+2)\uparrow p, n+1)$ is minimalizable.

Theorem: A function $f : \mathbb{N}^k \to \mathbb{N}$ is μ-recursive iff it is recursive (that is, computable by a TM).

Proof: ($\Rightarrow$)

If $f : \mathbb{N}^k \to \mathbb{N}$ is μ-recursive.

$\Rightarrow$ f is Turing Machine computable.

- f is μ-recursive, then it is defined from the basic functions by applications of composition, recursive definition, and minimalization on minimalizable functions.

- The basic functions are recursive.

- The composition and the recursive definition of recursive functions and the minimalization of minimalizable recursive functions are recursive.

$\Rightarrow$ all μ-recursive functions are recursive.

Remark:

To prove the above results, we use the random access Turing machine(equivalence with standard TM).

($\Leftarrow$) Suppose that a TM $M = (K, \sum, \delta, s, \{h\})$ computes $f : \mathbb{N} \to \mathbb{N}$.

$\Rightarrow$ f is μ-recursive.

- Without loss of generality that $K \cap \sum = \emptyset$. Let $b = |\sum| + |K|$, and map $\mathbb{E} : \sum \cup K \to \{0, 1, \ldots, b-1\}$, such that $\mathbb{E}(0) = 0$ and $\mathbb{E}(1) = 1$.

- The configuration $(q, a_1 a_2 \ldots \underline{a_k} \ldots a_n)$ will be represented as the base-b integer $a_1 a_2 \ldots a_k \underline{q} \ldots a_n$, as the integer

$$\mathbb{E}(a_1)b^n + \cdots + \mathbb{E}(a_k)b^{n-k+1} + \mathbb{E}(q)b^{n-k} + \mathbb{E}(a_{k+1})b^{n-k-1} + \cdots + \mathbb{E}(a_n)$$

• We will definite f as a μ-recursive function. Ultimately, f will be defined as

$$f(n) = num(output(last(comp(n)))).$$

– num is a function that takes an integer whose base-b representation is a string of 0's and 1's

– $output$ takes the integer represenating in base b a halted configuration of the form $\triangleright \sqcup hw$ and omits the first three symbols $\triangleright \sqcup h$.

– $comp(n)$ is the number whose representation in base b is the juxt a position of the unique sequence of the configurations that starts with $\triangleright \sqcup sw$, where w is the binary encoding of n, and ends with $\triangleright \sqcup hw'$, where w' is the bianry encoding of $f(n)$; such sequence exists, since we assuming that M computes a function f.

– $last$ takes an integer representing the juxtaposition of configurations, and extracts the last configuration in the sequence.

- We have to define all these functions:

$$lastpos(n) = \mu\ m[equal(digit(m,n,b), \mathbb{E}(\triangleright))\ \text{or}\ equal(m,n)]$$

$$output(n) = rem(n, b \uparrow \log(b \sim 2, n \sim 1) \sim 2)$$

$$num(n) = digit(1,n,b) \cdot 2 + digit(2,n,b) \cdot 2 \uparrow 2 + \cdots$$
$$+ digit(\log(b \sim 2, n \sim 1), n, b) \cdot 2 \uparrow \log(b \sim 2, n \sim 1)$$

$$comp(n) = \mu\ m[iscomp(mn)\ \text{and}\ halted(last(m))]$$

It follows that f is indeed a μ-recursive function.

Homework 4:	
P190	4.1.2(a)(b) 4.1.7 4.1.10
P200	4.2.2 (a)(b)(d) 4.2.4(a)(b)(c)
P232	4.6.2 (a)
P242	4.7.2 (a)(b)