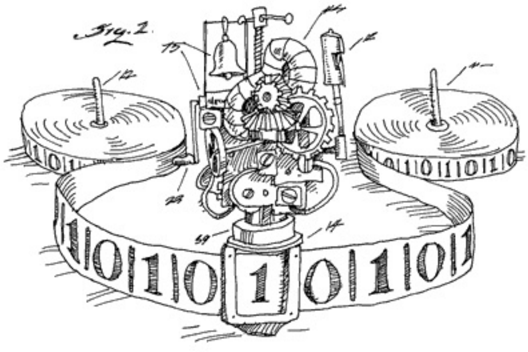
Automata, Languages and Computation

Chapter 8 : Turing Machines

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Lecture based on material originally developed by :
Gösta Grahne, Concordia University

Turing machines



1. Turing machine (TM) : formal model of computer algorithms that allows the mathematical study of computability
2. Programming techniques for TM : techniques to facilitate the writing of programs for TM
3. TM Extensions : machines that are more complex than TM but with the same computational capacity
4. TM with restrictions : automata that are simpler than TM but with the same computational capacity

Turing machine

In order to mathematically study undecidabilty we need a **simple** formalism to represent programs (Python is **not** suitable)

Historically used formalisms:

- predicate calculus (Gödel, 1931)
- partial recursive functions (Kleene, 1936)
- lambda calculus (Church, 1936)
- Turing machine (Turing, 1936)

Turing machine

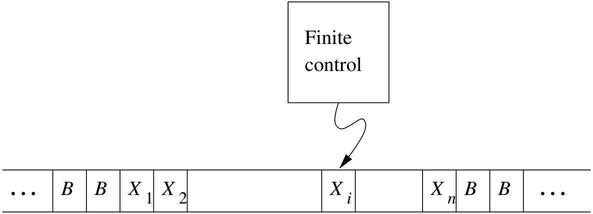
A Turing machine is a finite state automaton with the addition of a **memory tape** with

- sequential access
- unlimited capacity in both tape directions

Differently from the PDA model, input string is initially placed into the auxiliary memory

The Turing machine model allows the study of computability properties such as **undecidabilty** and **intractability**

Turing machine



Informally, a Turing machine performs a move according to its state and the symbol which is read by the tape head

In a single move, a Turing machine

- changes its state
- writes a new symbol in the cell read by the tape head
- moves the tape head to the cell to the right or to the left

Turing machine

A **Turing machine**, MT for short, is a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where

- Q is a finite set of **states**
- Σ is a finite set of **input symbols**
- Γ is a finite set of **tape symbols**, with $\Sigma \subseteq \Gamma$
- δ is a **transition function** from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$
- q_0 is the **initial state**
- $B \in \Gamma$ is the **blank symbol**, with $B \notin \Sigma$
- $F \subseteq Q$ is the set of **final states**

Note that the automaton is deterministic, and it has no 'stand' move

Instantaneous descriptions

A TM changes its configuration with each move. We use the notion of instantaneous description (ID) to describe configurations

An **instantaneous description** (ID) of M is a string of the form

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n$$

where

- q is M's state
- $X_1 X_2 \cdots X_n$ is the "visited" portion of M's tape
- the tape head of M is reading the i-th tape symbol

Computation of a TM

To represent a **computation step** of M we use the binary relation $\underset{M}{\vdash}$ defined on the set of IDs

If $\delta(q, X_i) = (p, Y, L)$, then

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \underset{M}{\vdash} X_1 X_2 \cdots p X_{i-1} Y X_{i+1} \cdots X_n$$

If $\delta(q, X_i) = (p, Y, R)$, then

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \underset{M}{\vdash} X_1 X_2 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$$

Special cases if the tape head is at the two ends of the written tape

Computation of a TM

To represent the **computations** of M, we use the reflexive and transitive closure of $\underset{M}{\vdash}$, written $\underset{M}{\overset{*}{\vdash}}$

For input string $w \in \Sigma^*$, the initial ID is $q_0 w$

For a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, an accepting computation has the form

$$q_0 w \underset{M}{\overset{*}{\vdash}} \alpha p \beta$$

with $p \in F$ and $\alpha, \beta \in \Gamma^*$

We will come back to this definition after some examples

Example

Let us specify a TM M with $L(M) = \{0^n 1^n \mid n \geqslant 1\}$

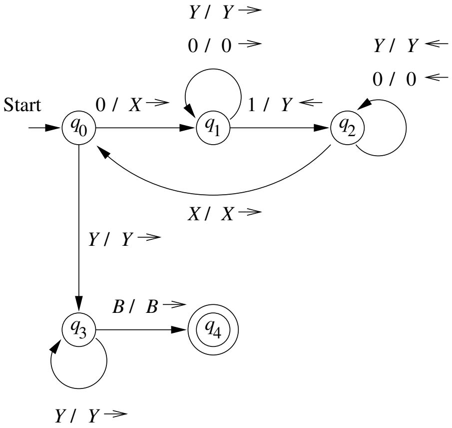
$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

The transition function δ is represented by the following table

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
$\star q_4$					

Example

We can also represent δ by means of the following **transition diagram**



Example

If input w has the form $\mathbf{0}^*\mathbf{1}^*$, then at each ID the tape is of the form $\boldsymbol{X}^*\mathbf{0}^*\boldsymbol{Y}^*\mathbf{1}^*$

M implements the following strategy

- in q_0 it replaces the leftmost 0 with X and moves to q_1
- in q_1 it proceeds from left to right, goes over 0 and Y looking for the leftmost 1, replaces it with Y and moves to q_2
- in q_2 it proceeds from right to left, goes over Y and 0 looking for the rightmost X, and moves back to q_0
- in q_0, if it finds one more 0 it resumes the above cycle, otherwise it moves to q_3
- in q_3 it overrides all of the Y's and accepts if there is no 1

Observe how input string is overwritten during the computation

Example

Given the string input 0011, M performs the following computation (sequence of ID)

$$
\begin{aligned}
q_0 0011 &\vdash Xq_1 011 \vdash X0q_1 11 \\
&\vdash Xq_2 0Y1 \vdash q_2 X0Y1 \\
&\vdash Xq_0 0Y1 \vdash XXq_1 Y1 \\
&\vdash XXYq_1 1 \vdash XXq_2 YY \\
&\vdash Xq_2 XYY \vdash XXq_0 YY \\
&\vdash XXYq_3 Y \vdash XXYYq_3 B \\
&\vdash XXYYBq_4 B
\end{aligned}
$$

TM with "output"

We have defined a TM as a recognition device. Alternatively, we can use these devices to compute **functions** on natural numbers.

Historically, this was the original definition by A. Turing

We encode each natural number in **unary notation** according to the scheme

$$n =_1 0^n$$

Example

The following TM M computes the **proper subtractor** function

$$m \dot{-} n = max(m - n, 0)$$

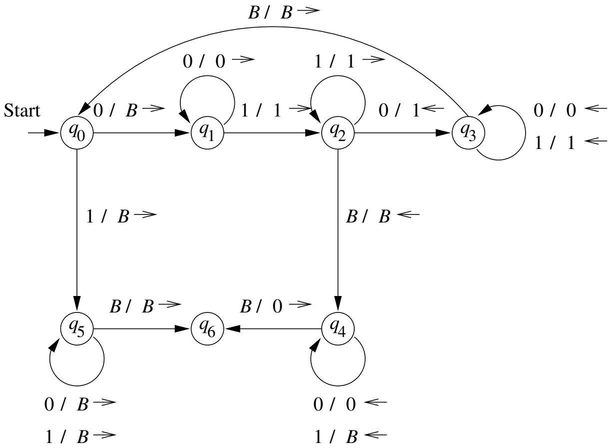
starting with $0^m 1 0^n$ on its tape and **halting** with $0^{m \dot{-} n}$.

No set of final states for TMs with output

	0	1	B
$\rightarrow q_0$	(q_1, B, R)	(q_5, B, R)	
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
$\star q_6$			

Example

The transition diagram is



Example

The TM M performs the following loop

- find the leftmost 0 and replace with B (states q_0, q_3)
- search right for the first 0 placed after symbols 1, and replace it with 1 (states q_1, q_2)

The loop ends in two possible ways

- M cannot find a 0 to the right of the 1's ($m > n$); then M turns all of the 1's into a single 0 followed by B's
- M cannot find a 0 to be replaced by B ($m \leqslant n$); then $m \dot{-} n = 0$ and M replaces all 0's and 1's into B

Notation for TM

We use notational **conventions** similar to those of other automata

- a, b, c, ..., a_1, a_2, ..., a_i, ... input symbols
- X, Y, Z tape symbols
- u, w, x, y, z strings over the input alphabet
- α, β, γ, ..., strings over tape alphabet
- p, q, r, ..., q_1, q_2, ..., q_i, ... states

Exercise

The TM $M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_2\})$ has the following transitions :

$$\delta(q_0, 0) = (q_1, 1, R)$$
$$\delta(q_1, 1) = (q_2, 0, L)$$
$$\delta(q_2, 1) = (q_0, 1, R)$$

Specify the computation (ID sequence) of M for input 0100

Exercise

Provide TMs for the following languages by specifying the transition diagram and by briefly explaining the adopted strategy

- $L = \{a^n b^{2n} \mid n \geqslant 1\}$
- $L = \{w \in \{a, b, c\}^* \mid \#_a(w) = \#_b(w) = \#_c(w)\}$
- $L = \{a^n b^{2k} a^n \mid b, k \geqslant 0\}$

Language accepted by a TM

A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ **accepts** the language

$$L(M) = \{w \mid w \in \Sigma^*,\ q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta,\ p \in F,\ \alpha, \beta \in \Gamma^*\}$$

The class of languages accepted by TMs is called **recursively enumerable** (RE)

This term derives from formalisms that historically preceded TM

TM and halting

A TM **halts** if it enters a state q with tape symbol X and $\delta(q, X)$ is not defined (there is no next move)

If a TM accepts a string, we can assume that it always halts : just make $\delta(q, X)$ undefined for every final state q

If a TM does not accept, **we can't** assume that it will halt (in a non-final state)

The class of languages accepted by some TM that halts for every input are called **recursive** (REC)

Recursive and recursively enumerable languages

Recursive language (REC) : the language is accepted by a TM that halts on each input string (in the language or not)

Recursively enumerable language (RE) : the language is accepted by a TM that halts when the string belongs to the language

For strings not in the language, the TM may compute forever

A decision problem P is **decidable** if its encoding L_P (see chapter 1) is a recursive language. Alternatively : if there is a TM M that always halts such that $L(M) = L_P$

Programming techniques for TM

Although the class of TM is very simple, this model has the **same** computational power as a modern computer

We will also see that a TM is able to perform processing on other TMs. This allows us to prove that certain problems are undecidable

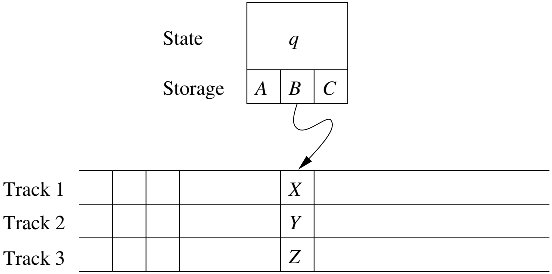
Compare with compilers, which take programs as input and produce new programs as output

We present in the following some **notational variants** of the TM that make TM programming easier

Programming techniques for TM

We reformulate the TM definition using

- a finite number of registers with **random access**, which we place inside each state
- a finite number of tape tracks



State as internal memory

Example : A TM M that "memorizes" the first symbol read and verifies that this does not appear again in the input

$$L(M) = L(01^* + 10^*)$$

Let $M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0, B], B, \{[q_1, B]\})$, with $Q = \{q_0, q_1\} \times \{0, 1, B\}$

	0	1	B
$\rightarrow [q_0, B]$	$([q_1, 0], 0, R)$	$([q_1, 1], 1, R)$	
$[q_1, 0]$		$([q_1, 0], 1, R)$	$([q_1, B], B, R)$
$[q_1, 1]$	$([q_1, 1], 0, R)$		$([q_1, B], B, R)$
$\star[q_1, B]$			

Tape with multiple tracks

Example : A TM for the language $L = \{wcw \mid w \in \{0,1\}^*\}$

We use a tape track for "marking" those input symbols that we have already tested

$$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_0, B]\})$$

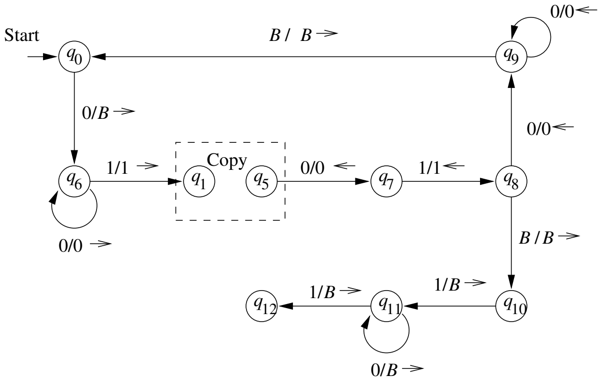
where

- $Q = \{q_1, q_2, \dots, q_9\} \times \{0, 1, B\}$
- $\Sigma = \{[B, 0], [B, 1], [B, c]\}$
- $\Gamma = \{B, *\} \times \{0, 1, c, B\}$

See the textbook for the specification of the transition function δ

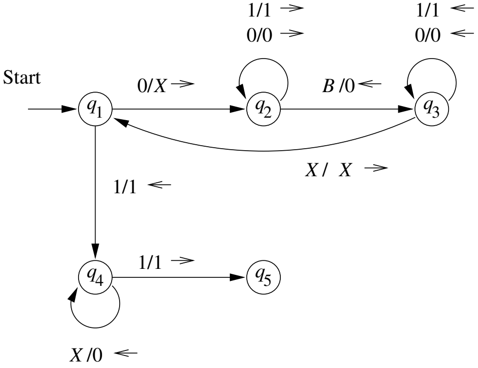
Use of a subroutine

Example : A TM for the computation of the product function $0^m10^n1 \mapsto 0^{m \cdot n}$. We use a *subroutine* "Copy"



Use of a subroutine

The subroutine "`Copy`" takes ID $0^{m-k}1q_1 0^n 1 0^{(k-1)n}$ to ID $0^{m-k}1q_5 0^n 1 0^{kn}$



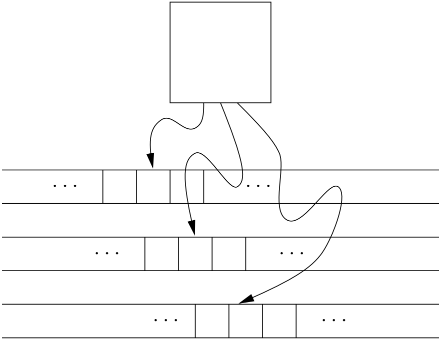
TM extensions

Let us now present some **extensions** of the TM definition

For each extension, we prove that the **computational capacity** is the same as the one of the classic definition of TM

Multi-tape TM

We use a finite number of **independent** tapes for the computation, with the input on the first tape



Multi-tape TM

In a single move the multi-tape TM performs the following actions

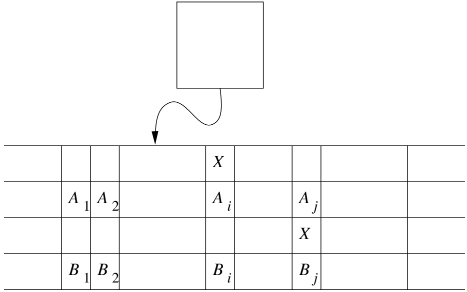
- state update, on the basis of read tape symbols
- for each tape :
  - write a symbol in current cell
  - move the tape head independently of the other heads (L = left, R = right, or S = stay)

Note that the stay option is not available in a TM

Multi-tape TM

Theorem A language accepted by a multi-tape TM M is RE

Proof (sketch) We can simulate M using a TM N with a multi-track tape



Multi-tape TM

We use $2k$ tracks to simulate k tapes : even tracks used for tape content, odd tracks used for tape head position

N visits all k head positions to **simulate** a single move of M

- left to right pass : the number of visited tape heads and the content of the corresponding cells are stored into the state of N
- right to left pass : for each tape head of M, the corresponding action is simulated by N

N updates its state in the same way as M $\square$

Multi-tape TM

Theorem The TM N in the proof of the previous theorem simulates the first n moves of the TM M with k tapes in time $\mathcal{O}(n^2)$

Proof (sketch) After n moves of M, tape head markers in N have **mutual distance** not exceeding $2n$

It follows that any one of the first n moves of M can be simulated by N in a number of moves not exceeding $4n + 2k$, which amounts to $\mathcal{O}(n)$ since k is a constant $\square$

Nondeterministic TM

In a **nondeterministic** Turing machine, NTM for short, the transition function δ is set-valued :

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$

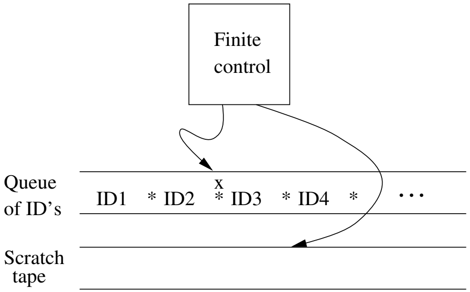
At each step, the NTM chooses one of the triples as the next move

The NTM accepts an input w if **there exists** a sequence of choices that leads from the initial ID for w to an ID with an accepting state

Nondeterministic TM

Theorem For each NTM M_N, there exists a (deterministic) TM M_D such that $L(M_N) = L(M_D)$

Proof (skecth) We specify M_D as a TM with two tapes



Nondeterministic TM

A single ID in the **queue** (first) tape is marked as being processed

M_D performs the following cycle

- copy the marked ID from the queue tape to the **scratch** (second) tape
- for each possible move of M_N, add a new ID at the end of queue tape
- move the marker in the queue tape to the next ID

Nondeterministic TM

Let m be the maximum number of choices for M_N. After n moves, M_N reaches a number of ID bounded by

$$1 + m + m^2 + \cdots + m^n \leqslant nm^n + 1$$

M_D explores all the IDs reached by M_N in n steps before each ID reached in $n + 1$ steps, as in a **breadth first** search

If there exists an accepting ID for M_N on w, M_D reaches this ID in a finite amount of time. Otherwise, M_D does not accept, and may not halt

We therefore conclude that $L(M_N) = L(M_D)$ $\square$

Nondeterministic TM

Observe that the TM M_D in the previous theorem can take an amount of time **exponentially larger** than M_N to accept an input string

We **do not know** if this slowdown is necessary: this very important issue will be the subject of investigation in a next chapter

TM with restrictions

We impose some restrictions on the definition of TM / multi-tape TM:

- tape is unlimited only in one direction
- two tapes used in stack mode

We prove that these models are equivalent to TM

Think about the above definitions as normal forms

These models are especially useful in some proofs that we will present later on

TM with semi-infinite tape

In a TM with **semi-infinite tape**

- there are no cells to the left of the **initial tape position**
- a tape symbol can never be overwritten by the blank B

In a TM with semi-infinite tape each ID is a sequence of tape symbols other than B, i.e., there are no "holes"

TM with semi-infinite tape

We can **simulate** a TM by means of a TM with semi-infinite tape with two tracks

- the upper track represents the initial position X_0 and all tape cells to its right
- the lower track represents all tape cells to the left of X_0, in reverse order
- a special symbol $*$ is used to mark the initial position

X_0	X_1	X_2	$\ldots$
$*$	X_{-1}	X_{-2}	$\ldots$

TM with semi-infinite tape

Theorem Each language accepted by a TM M_2 is also accepted by a TM M_1 with semi-infinite tape

Proof (sketch) First, we modify M_2 in such a way that it uses a **new** tape symbol B' each time B is used to overwrite a tape symbol

Let $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, B, F_2)$ be the modified TM. We define

$$M_1 = (Q_1, \Sigma \times \{B\}, \Gamma_1, \delta_1, q_0, [B, B], F_1)$$

TM with semi-infinite tape

The states of M_1 are $Q_1 = \{q_0, q_1\} \cup (Q_2 \times \{U, L\})$. Symbols U, L indicate whether M_1 is visiting the upper or lower track

The input symbols of M_1 are pairs $[a, B]$ with a an input symbol of M_2

The tape symbols Γ_1 of M_1 are pairs in $\Gamma_2 \times \Gamma_2$ with the addition of pairs $[X, *]$ for each $X \in \Gamma_2$, where $*$ is used to mark the initial position of M_1 tape

The accepting symbols of M_1 are $F_1 = F_2 \times \{U, L\}$

TM with semi-infinite tape

Transitions in δ_1 implement the following moves

- place $*$ on the initial position, in the lower track, and restore the initial conditions of M_2
- when M_1 is not in the initial cell, the moves of M_2 are simulated with
  - the same direction if U appears in the state
  - the reverse direction if L appears in the state
- upon reading $*$
  - if M_2 moves to the right, M_1 simulates the same move
  - if M_2 moves to the left, M_1 simulates the same move but it reverses the direction

TM with semi-infinite tape

It can be shown by induction on the number of steps of a computation that the IDs of M_1 and M_2 match, modulo

- the reversal of the L track of M_1
- its concatenation on the left with the U track of M_1
- the elimination of the $*$ marker

It follows that $L(M_1) = L(M_2)$ □

Multi-Stack machine

We apply to a multi-tape TM the restriction to use each tape in stack mode

- can only overwrite at the top
- can only insert at the top
- can only delete at the top

The resulting model accepts only recursively enumerable language, since it is a restriction of a multi tape TM

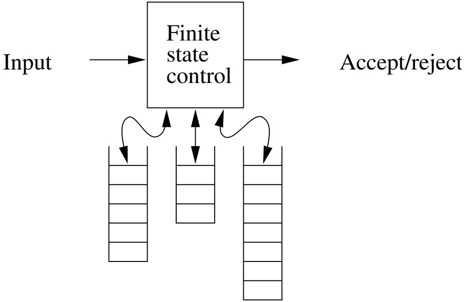
Multi-Stack machine

Let M be a multi-tape TM with tapes used in stack mode. We also assume that

- the input is provided in an **external**, read-only tape and with end-marker \$, and it can only be read from left to right
- M can perform ϵ-moves, but these moves must not be in conflict with each other or with other reading moves (determinism)

Multi-Stack machine

M is called a **multi-stack machine**, and can be viewed as a generalization of the deterministic PDA



Multi-Stack machine

In a multi-stack machine with k stacks, a **transition rule** has the form

$$\delta(q, a, X_1, X_2, \ldots, X_k) = (p, \gamma_1, \gamma_2, \ldots, \gamma_k)$$

In words, when the machine is in state q and reads input symbol $a \in \Sigma \cup \{\epsilon\}$, and with X_i on top of the i-th stack, $1 \leqslant i \leqslant k$, it moves to state p and replaces each X_i with γ_i

Multi-Stack machine

Theorem If a language L is accepted by a TM, then L is accepted by a multi-stack machine with two stacks

Proof (sketch) Let $L = L(M)$ for a TM M. We construct a machine S with two stacks, having special symbols used as **markers** at the bottom of the stack

The basic idea is to

- simulate the tape to the left of the current position with the first stack
- simulate the tape starting from the current position and extending to the right with the second stack

Multi-Stack machine

The transition rules of S implement the following strategy

- copy the input $w\$$ into the first stack
- move the contents of the first stack into the second stack
- if M overwrites X with Y and moves to the right, S pushes Y on the first stack and pops X from the second stack
- if M overwrites X with Y and moves to the left, S pops the symbol Z from the first stack and replaces X with ZY in the second stack
- in addition, S employs some special moves to handle the case where M is located at the end points of the tape (one of the two stacks contains the bottom marker)
- S accepts whenever M accepts □

TM and computer

Theorem If a language L is accepted by a modern computer, then L is accepted by a TM

Proof Omitted