We will now convince ourselves that Turing machine can accomplish any task that a "real", "conventional" computer can. One way to see this is to note that

C-Program $\overset{\text{Equivalent to}}{\equiv}$ Assembly language program

$\overset{\text{Equivalent to}}{\equiv}$ Turing Macine program.

An assembly language program operates with

- constantly many registers
- Memory
- Constantly many instructions (= elementary operations) such as moving/adding content of one register to another, reading/writing from memory, etc.

It can now be observed that registers, memory, and the basic instructions can all be simulated by a TM!

To further make this point, let's see that some basic tasks can be executed on a TM.

String Matching For example, a TM can decide

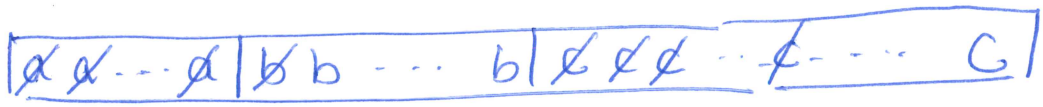
$$L = \{ x \# y \mid x, y \in \{0,1\}^*, \; x = y \},$$

i.e. matching x and y.

Addition For example, a TM can decide

$$L = \{ a^i b^j c^k \mid i + j = k, \; i, j, k \geq 0 \}.$$

The m/c first checks that the input is of the form $a^* b^* c^*$. Then

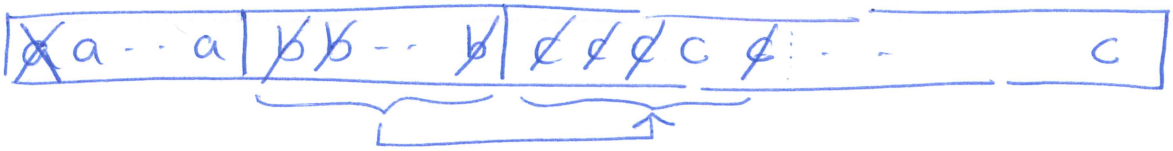


the m/c shuttles back-and-forth and crosses out 'a's and 'b's and crosses out one 'c' for every 'a' or 'b' crossed out. It accepts iff all 'a', 'b', 'c's get crossed out in this manner.

Multiplication For example, a TM can decide

$$L = \{ a^i b^j c^k \mid i \cdot j = k, \; i, j, k \geq 1 \}.$$

The TM



- Crosses out 'a'.
- Crosses out one 'c' for every 'b' (which is also crossed out).
- Goes back and restores all 'b's.
- Repeats these steps with the next 'a', until all 'a's are crossed out.

Accepts iff all 'a's and 'c's get crossed out in this manner.

The addition and multiplication as described operate on integers expressed in unary. It is certainly possible to

switch between unary and binary represent^{n}!

Example A TM can decide

$$L = \left\{ a^n \# x \;\middle|\; \begin{array}{l} x \in \{0,1\}^*, \; x \text{ represents} \\ n \text{ in binary} \end{array} \right\}$$

Idea is to cross out every alternate 'a', starting with the first one, e.g.

$\not{a}a\not{a}a\not{a}a\not{a}a\not{a}$.

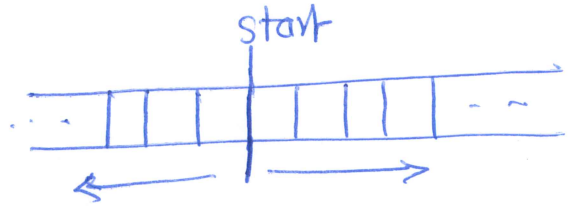
Depending on whether the last 'a' gets crossed out or not, the number of 'a's is odd or even, and hence the last bit of x is '1' or '0'; this can be checked and the last bit of x is crossed out. The procedure is now repeated with the 'a's that have survived and with bits of x from the end. ▨

With these examples and with the analogy to assembly lang. program, we hope that

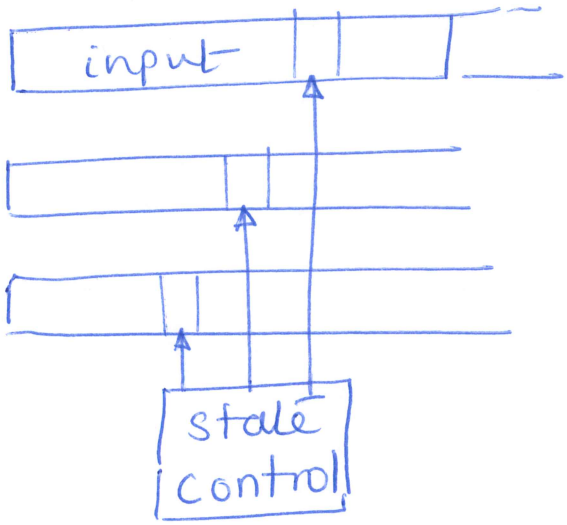
the reader is convinced that a TM can "do" everything a "real" computer can do!

It is possible to enhance the basic TM model to allow

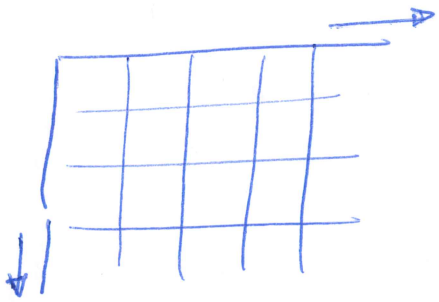
- 2-way infinite tape



- More than one tapes



- 2-dimensional tape/memory



However these enhancements do not really add much power, i.e. the basic TM model can simulate them all (with some "acceptable" loss of efficiency).

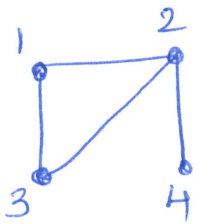
Encodings, Computational Problems, Algorithms

We recall that combinatorial objects can be encoded as strings. An encoding of an object O is denoted as $\langle O \rangle$. The specific choice or manner of encoding is not important.

Encoding integers

$\langle n \rangle$ = Binary encoding of integer n.

Encoding graphs

G : 

$\rightarrow$

	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	0
4	0	1	0	0

Adjacency matrix

rows of the adj matrix.

$\langle G \rangle$ = 0110 # 1011 # 1100 # 0100

Encoding Polynomials (with integer coefficients)

$\langle P \rangle = \langle n \rangle \# \langle d_1 \rangle : \langle d_2 \rangle : \cdots : \langle d_n \rangle$

$\# \text{Coeff}, \text{Coeff}, \cdots, \text{Coeff} \#$

where n = number of variables

$d_1, \cdots, d_n$ = max degrees in these variables.

"Coeff" = integer coefficients of all monomials listed in some canonical order.

A very interesting fact is that computational models (e.g. finite automata, PDAs, and TMs!) can be encoded as well.

Encoding Finite Automata

$$M = (Q, \Sigma, q_1, \delta, F)$$

Suppose $|Q| = n$, $|\Sigma| = \ell$.

The states can be taken as $1, 2, 3, \ldots, n$.

The input symbols " $1, 2, 3, \ldots, \ell$.

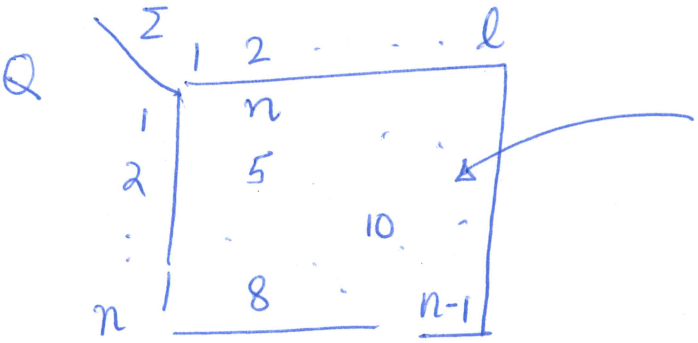
State numbered 1 is the start state by convention.

The set $F \subseteq Q$ can be represented by a bit string $\langle F \rangle \in \{0,1\}^n$, whose i^{th} bit is '1' or '0' depending on whether State i is an accept state or not. The FA can now be encoded as :

$\langle M \rangle = \langle n \rangle \# \langle \ell \rangle \# \langle F \rangle \# \langle \delta \rangle \#$

Here $\langle \delta \rangle$ will be the encoding of the transition function $\delta : Q \times \Sigma \to Q$.

Since $Q = \{1, 2, \ldots, n\}$, $\Sigma = \{1, 2, \ldots, \ell\}$



$n \times \ell$ matrix with integer entries between 1 thru n.

$\langle \delta \rangle$ = $n \cdot \ell$ integers listed in some canonical order (e.g. row by row).

Encoding Grammars $G = (V, \Sigma, S, R)$

Suppose a grammar has n variables $|V| = n$
ℓ terminals. $|\Sigma| = \ell$

Variables can be denoted by integers $1, 2, \ldots, n$

Terminals " $n+1, \ldots, n+\ell$.

A rule $A \to W$, $W \in (V \cup \Sigma)^*$ can be naturally encoded by encoding A and symbols in W by their integer representatives.

The start variable is numbered 1 by convention.

Provided there are m rules,

$$\langle G \rangle = \langle n \rangle \# \langle l \rangle \# \langle rule_1 \rangle : \langle rule_2 \rangle : \cdots : \langle rule_m \rangle \#$$

Finally, we note a most important fact that Turing m/c descriptions (= programs) can be encoded as strings.

Encoding TMs

$$M = (Q, \Sigma, \Gamma, q_{start}, q_{accept}, q_{reject}, \delta)$$

Suppose $|Q| = n$, $|\Sigma| = l$, $|\Gamma| = k$,

$\Sigma \subseteq \Gamma$, $'\sqcup' \in \Gamma \setminus \Sigma$.

States in Q are represented by integers $1, 2, \cdots, n$

Symbols in Σ " $1, 2, \cdots, l$

Symbols in $\Gamma \setminus \Sigma$ " $l+1, \cdots, k$.

Symbol '$\sqcup$' is numbered $l+1$ by convention.

States $q_{start}, q_{accept}, q_{reject}$ are numbered 1, 2, 3 by convention.

Note that transition function δ is

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}.$$

It can be encoded by $n \cdot k$ matrix with entries of the type (i, j, L), $1 \le i \le n$, $1 \le j \le k$ or (i, j, R).

$\langle \delta \rangle$ = List of the nk entries of this matrix (say row by row).

Finally, the TM encoding (= description) is

$$\langle M \rangle = \langle n \rangle \# \langle l \rangle \# \langle k \rangle \# \langle \delta \rangle \# .$$

Thanks to the encodings, decision problems can be phrased as language recognition problems.

For example, (problems and corresponding lang.)

① Is given graph connected ? $L_{conn} = \{\langle G \rangle \mid G \text{ is a connected graph.}\}$

② Is given integer a prime ? $L_{prime} = \{\langle n \rangle \mid n \text{ is a prime}\}$.

Interestingly, one can pose decision problems about computational models. For example,

③ Does a given DFA accept at least one input string?

$$E_{DFA} = \{ \langle D \rangle \mid \begin{array}{l} D \text{ is a DFA} \\ \text{s.t. } L(D) \neq \phi \end{array} \}$$

④ Does a given c.f. grammar generate at least one string?

$$E_{CFG} = \{ \langle G \rangle \mid \begin{array}{l} G \text{ is a c.f.g.} \\ \text{s.t. } L(G) \neq \phi \end{array} \}$$

⑤ Is a given code a syntactically correct C-program?

$$L_{program} = \{ \langle C \rangle \mid \begin{array}{l} C \text{ is a} \\ \text{syn. correct} \\ \text{C-program} \end{array} \}.$$

One can phrase Problem ⑤ also in terms of Turing machines (which are equivalent to C-programs!)

⑤' Is a given string a valid encoding of a Turing m/c?

$$L_{TM} = \{ \langle M \rangle \mid \begin{array}{l} M \text{ is a} \\ \text{Turing m/c} \end{array} \}$$

All the problems/languages above ①-⑤, ⑤' are decidable. We can also say that they can be solved by an "algorithm".

Def An algorithm is a TM that halts on every input and gives the correct answer.

However, it turns out that some problems are undecidable, i.e. there is no algorithm that solves them!

Examples of undecidable problems/languages (though we will not prove their undecidability in all cases):

① Does a given TM accept at least one string?

$$E_{TM} = \{\langle M \rangle \mid \begin{array}{l} M \text{ is a TM} \\ \text{s.t. } L(M) \neq \phi \end{array}\}$$

② Halting Problem on ε.
Does a given TM halt on input ε.

$$HALT_{TM}^{\varepsilon} = \{\langle M \rangle \mid \begin{array}{l} M \text{ is a TM} \\ \text{s.t. } M \text{ halts} \\ \text{on input } \varepsilon \end{array}\}$$

③ Does a given cf. grammar generate all strings?

$$A\ll_{CFG} = \{\langle G\rangle \mid G \text{ is a cfg s.t. } L(G) = \Sigma^*\}.$$

④ Does a given polynomial with integer coefficients have an integer root?

$$L_{Hilbert} = \{\langle P\rangle \mid P \text{ is a polynomial with integer root}\}.$$

Problem ④ is known as Hilbert's 10^{th} Problem. It turns out to be undecidable. To clarify, here one is talking about polynomials in multiple variables and arbitrary degree, e.g.

$$6x^2y - 9xyz + y^3 + z^3 - 156.$$

If does have a root, e.g. $x=2$, $y=5$, $z=1$ that makes the polynomial evaluate to Zero.

In this course, we will indeed show that Problems ①, ② are undecidable.

Problems ③, ④, and many more are then shown to be undecidable by a reduction from (say) Problem ②. We will show undecidability of Problem ③ if time permits.

Church-Turing Thesis

Our intuitive notion of an "algorithm" or "solvability of a problem" (seemingly) corresponds to Turing machine programs. So are TMs a "universal" model of computation? Yes, it is believed so. Church and Turing stated the hypothesis that

> "Any physically realizable computational device can be simulated by (standard) TM up to a polynomial loss in efficiency".

So far, this thesis has been holding up correct, but recently the model of

Quantum computer is presenting a challenge. It is still an open question to settle this matter one way or the other; we will not pursue this topic here.

To clarify the phrase "polynomial loss of efficiency", the phrase indicates that if a problem is solved in time T ($= T(n)$ as a function of input size n) on a physically realizable device, then there is a simulation on a TM that runs in time polynomial in T, i.e. T, T^2, T^3, etc.

This is necessary. For example, recognizing palindromes on (Standard 1-tape) TM takes time n^2 whereas on a 2-tape TM, it can be done in time n (how?).

Polynomial loss in time efficiency is considered "acceptable". We will see later why.