

# Computability and Complexity Theory

## Computability and complexity

- *Computability theory*
  - What problems can be solved on a computer ?
  - What is a computable function ?
  - Decidable vs. undecidable problems
- *Complexity theory*
  - How much time and memory is needed to solve a problem ?
  - Tractable vs. intractable problems

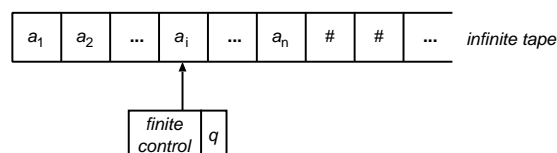
## What is a computable function ?

- Non-trivial question  $\rightsquigarrow$  various formalizations, e.g.
  - General recursive functions *Gödel/Herbrand/Kleene 1936*
  - $\lambda$ -calculus *Church 1936*
  - $\mu$ -recursive functions *Gödel/Kleene 1936*
  - Turing machines *Turing 1936*
  - Post systems *Post 1943*
  - Markov algorithms *Markov 1951*
  - Unlimited register machines *Shepherdson-Sturgis 1963*
  - ...
- All these approaches have turned out to be equivalent.

## Church's thesis

The class of intuitively computable functions is equal to the class of Turing computable functions.

## Turing machine



Depending on the symbol scanned and the state of the control, in each step the machine

- changes state,
- prints a symbol on the cell scanned, replacing what is written there,
- moves the head left or right one cell.

## Formal definition

- $M = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$
- $Q$  is the finite set of *states*.
- $\Gamma$  is the finite alphabet of allowable *tape symbols*.
- $\# \in \Gamma$  is the *blank*.
- $\Sigma \subset \Gamma \setminus \{\#\}$  is the set of *input symbols*.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  is the *next move function* (possibly undefined for some arguments)
- $q_0 \in Q$  is the *start state*.
- $F \subseteq Q$  is the set of *final (accepting) states*.

## Recognizing languages

- *Instantaneous description*:  $\alpha_l q \alpha_r$ , where
  - $q$  is the current state,
  - $\alpha_l \alpha_r \in \Gamma^*$  is the string on the tape up to the rightmost nonblank symbol,
  - the head is scanning the leftmost symbol of  $\alpha_r$ .
- *Move*:  $\alpha_l q \alpha_r \vdash \alpha'_l q' \alpha'_r$ , by one step of the machine.
- *Language accepted*

$$L(M) = \{w \in \Sigma^* \mid q_0 w \vdash^* \alpha_l q \alpha_r, \text{ for some } q \in F \text{ and } \alpha_l, \alpha_r \in \Gamma^*\}$$

- $M$  may not halt, if  $w$  is not accepted.

## Example

- *Turing machine*

$$M = (\{q_0, \dots, q_4\}, \{0, 1\}, \{0, 1, X, Y, \#\}, \delta, q_0, \#, \{q_4\})$$

accepting the language  $L = \{0^n 1^n \mid n \geq 1\}$

$\delta$	0	1	X	Y	#
$q_0$	$(q_1, X, R)$	–	–	$(q_3, Y, R)$	–
$q_1$	$(q_1, 0, R)$	$(q_2, Y, L)$	–	$(q_1, Y, R)$	–
$q_2$	$(q_2, 0, L)$	–	$(q_0, X, R)$	$(q_2, Y, L)$	–
$q_3$	–	–	–	$(q_3, Y, R)$	$(q_4, \#, R)$
$q_4$	–	–	–	–	–

- *Example computation*

$$\begin{array}{l}
 q_0 0 0 1 1 \vdash X q_1 0 1 1 \vdash X 0 q_1 1 1 \vdash X q_2 0 Y 1 \vdash \\
 q_2 X 0 Y 1 \vdash X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash \\
 X X q_2 Y Y \vdash X q_2 X Y Y \vdash X X q_0 Y Y \vdash X X Y q_3 Y \vdash \\
 X X Y Y q_3 \vdash X X Y Y \# q_4
 \end{array}$$

## Recursive languages

- A language  $L \subseteq \Sigma^*$  is *recursively enumerable* if  $L = L(M)$ , for some Turing machine  $M$ .

$$w \longrightarrow \boxed{M} \longrightarrow \begin{cases} \text{yes,} & \text{if } w \in L \\ \text{no,} & \text{if } w \notin L \\ M \text{ does not halt,} & \text{if } w \notin L \end{cases}$$

- A language  $L \subseteq \Sigma^*$  is *recursive* if  $L = L(M)$  for some Turing machine  $M$  that halts on all inputs  $w \in \Sigma^*$ .

$$w \longrightarrow \boxed{M} \longrightarrow \begin{cases} \text{yes,} & \text{if } w \in L \\ \text{no,} & \text{if } w \notin L \end{cases}$$

- **Lemma.**  $L$  is recursive iff both  $L$  and  $\bar{L} = \Sigma^* \setminus L$  are recursively enumerable.

## Enumerating languages

- An *enumerator* is a Turing machine  $M$  with extra output tape  $T$ , where symbols, once written, are never changed.
- $M$  writes to  $T$  words from  $\Sigma^*$ , separated by \$.
- Let  $G(M) = \{w \in \Sigma^* \mid w \text{ is written to } T\}$ .

## Some results

- **Lemma.** For any finite alphabet  $\Sigma$ , there exists a Turing machine that generates the words  $w \in \Sigma^*$  in *canonical ordering* (i.e.,  $w \prec w' \Leftrightarrow |w| < |w'|$  or  $|w| = |w'|$  and  $w \prec_{lex} w'$ ).
- **Lemma.** There exists a Turing machine that generates all pairs of natural numbers (in binary encoding).  
*Proof:* Use the ordering  $(0, 0), (1, 0), (0, 1), (2, 0), (1, 1), (0, 2), \dots$
- **Proposition.**  $L$  is recursively enumerable iff  $L = G(M)$ , for some Turing machine  $M$ .

## Computing functions

- Unary encoding of natural numbers:  $i \in \mathbb{N} \mapsto \underbrace{|| \dots ||}_{i \text{ times}} = |^i$   
(binary encoding would also be possible)

- $M$  computes  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  with  $f(i_1, \dots, i_k) = m$ :

- Start:  $|^{i_1} 0 |^{i_2} 0 \dots |^{i_k}$
- End:  $|^m$

- $f$  *partially recursive*:

$$i_1, \dots, i_k \longrightarrow \boxed{M} \longrightarrow \begin{cases} \text{halts with } f(i_1, \dots, i_k) = m, \\ \text{does not halt, i.e., } f \text{ undefined.} \end{cases}$$

- $f$  *recursive*:

$$i_1, \dots, i_k \longrightarrow \boxed{M} \longrightarrow \text{halts with } f(i_1, \dots, i_k) = m.$$