Computational Complexity; slides 2, HT 2019
Turing machines, undecidability

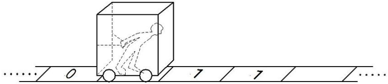
Prof. Paul W. Goldberg (Dept. of Computer Science,
University of Oxford)

HT 2019

Alan Turing considered qn. of "What is computation?" in 1936.

He argued, that *any* computation can be done using the following steps (writing on a sheet of paper):

- Concentrate on one part of the problem (one symbol on the paper)
- Depending on what you read there
  - Change into a new state (memorise a finite amount of information)
  - Modify this part of the problem
  - Move to another part of the input
- Repeat until finished



Deterministic Turing Machines

Definition: (one of many variants, all "equivalent")

A (deterministic) k-tape *Turing machine* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set of *states*
- Σ is *input alphabet* – a finite alphabet of *symbols*
- $\Gamma \supseteq \Sigma \cup \{\Box\}$ is *working tape alphabet* (finite)
- δ is the *transition function*
- $q_0 \in Q$ is the *initial state*
- $F \subseteq Q$ is a set of final states

Deterministic Turing Machines

Definition: (one of many variants, all "equivalent")

A (deterministic) k-tape *Turing machine* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set of *states*
- Σ is *input alphabet* – a finite alphabet of *symbols*
- $\Gamma \supseteq \Sigma \cup \{\square\}$ is *working tape alphabet* (finite)
- δ is the *transition function*
- $q_0 \in Q$ is the *initial state*
- $F \subseteq Q$ is a set of final states

Tapes:

Infinite tapes, bounded to the left.

Each cell contains one symbol from Γ ($\square$: special "blank" symbol)

Deterministic Turing Machines

Definition: (one of many variants, all "equivalent")

A (deterministic) k-tape *Turing machine* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a finite set of *states*
- Σ is *input alphabet* – a finite alphabet of *symbols*
- $\Gamma \supseteq \Sigma \cup \{\Box\}$ is *working tape alphabet* (finite)
- δ is the *transition function*
- $q_0 \in Q$ is the *initial state*
- $F \subseteq Q$ is a set of final states

Tapes:

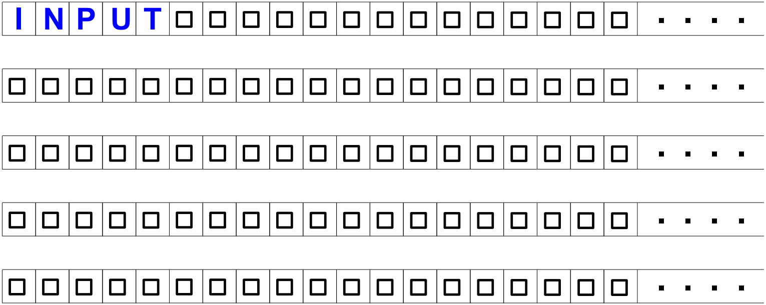
Infinite tapes, bounded to the left.

Each cell contains one symbol from Γ ($\Box$: special "blank" symbol)

I	**N**	**P**	**U**	**T**	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	$\Box$	. . . .

Transition function: $\delta : (Q \setminus F) \times \Gamma^k \to Q \times \Gamma^k \times \{-1, 0, 1\}^k$

(-1: "left" 0: "stay put" 1: "right")



Transition function: $\delta : (Q \setminus F) \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{-1, 0, 1\}^k$

(-1: "left" 0: "stay put" 1: "right")

Deterministic Turing Machines

Transition function: $\delta : (Q \setminus F) \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{-1, 0, 1\}^k$

(-1: "left" $\quad 0$: "stay put" $\quad 1$: "right")

Deterministic Turing Machines

Transition function: $\delta : (Q \setminus F) \times \Gamma^k \to Q \times \Gamma^k \times \{-1, 0, 1\}^k$

(-1: "left" 0: "stay put" 1: "right")

Transition function: $\delta : (Q \setminus F) \times \Gamma^k \to Q \times \Gamma^k \times \{-1, 0, 1\}^k$

(-1: "left" $\quad$ 0: "stay put" $\quad$ 1: "right")

Turing Machine operations

1. At each step of operation the machine is in one state $q \in Q$
2. Initially:
   - Machine is in state $q_0 \in Q$
   - the input is contained on tape 1
   - all other tape symbols are $\square$
3. The machine is reading one symbol on each tape: $s_1 \dots s_k$
4. To execute one step, the machine looks up

$$\delta(q, s_1, \dots, s_k) := \big(q', (s'_1, \dots, s'_k), (m_1, \dots, m_k)\big)$$

5. The machine:
   - changes to state q'
   - replaces each s_i by s'_i
   - moves the heads on the individual tapes according to m_i ($1 =$ move right, $-1 =$ move left, $0 =$ stay)
   - Execution stops when a final state is reached.
   - In this case, the content of the last tape k contains the output.

Example

What does the following 2-tape Turing machine do?

$$\mathcal{M} := \big(\{q_0, q_1, q_f\}, \{a, b\}, \{a, b, \Box\}, \delta, q_0, \{q_f\}\big)$$

where

$$\delta := \left\{ \begin{array}{l} \left(q_0, \binom{a}{-}, \binom{a}{-}, \binom{1}{0}, q_0\right) \\ \left(q_0, \binom{b}{-}, \binom{b}{-}, \binom{1}{0}, q_0\right) \\ \left(q_0, \binom{\Box}{-}, \binom{\Box}{-}, \binom{-1}{0}, q_1\right) \\ \left(q_1, \binom{a}{-}, \binom{\Box}{a}, \binom{-1}{1}, q_1\right) \\ \left(q_1, \binom{b}{-}, \binom{\Box}{b}, \binom{-1}{1}, q_1\right) \\ \left(q_1, \binom{\Box}{-}, \binom{\Box}{-}, \binom{0}{0}, q_f\right) \end{array} \right\}$$

Abbreviation $\binom{a}{-}$**:** $-$ stands for any symbol in Γ.

Configuration: A (k-tape) Turing machine $\mathcal{M} := (Q, \Sigma, \Gamma, \delta, q_0, F)$ in operation is completely described by

- the current state
- the contents of all its tapes (finite prefix that has been visited)
- the position of all its heads

$$\big(q,\ (w_1, \ldots, w_k),\ (p_1, \ldots, p_k)\big)$$

where $q \in Q$, $w_i \in \Gamma^*$, $p_i \in \mathbb{N}$

Configurations

Configuration: A (k-tape) Turing machine $\mathcal{M} := (Q, \Sigma, \Gamma, \delta, q_0, F)$ in operation is completely described by

- the current state
- the contents of all its tapes (finite prefix that has been visited)
- the position of all its heads

$$\big(q,\ (w_1, \ldots, w_k),\ (p_1, \ldots, p_k)\big)$$

where $q \in Q$, $w_i \in \Gamma^*$, $p_i \in \mathbb{N}$

Notation. Γ^*: set of words over the alphabet Γ

$$\Gamma^* := \{w := a_1 \ldots a_n : a_i \in \Gamma \text{ for all } 1 \le i \le n\}$$

We write ε for the empty word.

Configurations

Configuration: A (k-tape) Turing machine $\mathcal{M} := (Q, \Sigma, \Gamma, \delta, q_0, F)$ in operation is completely described by

- the current state
- the contents of all its tapes (finite prefix that has been visited)
- the position of all its heads

$$\big(q,\ (w_1, \dots, w_k),\ (p_1, \dots, p_k)\big)$$

where $q \in Q$, $w_i \in \Gamma^*$, $p_i \in \mathbb{N}$

Start configuration on input w: Triple $\big(q_0, (w, \varepsilon, \dots, \varepsilon), (0, \dots, 0)\big)$

q_0 initial state, tape 1 contains the input, all other tapes are empty, all heads on position 0 (ε : empty word)

Stop configuration:
Configuration $\big(q, (w_1, \dots, w_k), (p_1, \dots, p_k)\big)$ such that $q \in F$.

Computation

Notation:

- $C \vdash_{\mathcal{M}} C'$ if $\mathcal{M}$ can change from configuration C to C' in one step.
- $C \vdash^*_{\mathcal{M}} C'$ if $\mathcal{M}$ can change from configuration C to C' in arbitrarily many steps.

Computation

Notation:

- $C \vdash_{\mathcal{M}} C'$ if $\mathcal{M}$ can change from configuration C to C' in one step.
- $C \vdash^{*}_{\mathcal{M}} C'$ if $\mathcal{M}$ can change from configuration C to C' in arbitrarily many steps.

The *computation* of a TM $\mathcal{M}$ on input $w \in \Sigma^*$ is either

- an infinite sequence $C_0 \vdash_{\mathcal{M}} C_1 \vdash_{\mathcal{M}} C_2 \ldots$ of configurations or
- a finite sequence $C_0 \vdash_{\mathcal{M}} C_1 \vdash_{\mathcal{M}} C_2 \cdots \vdash_{\mathcal{M}} C_n$ of configurations.

  In the latter case we say that $\mathcal{M}$ *halts* on input w.

  Notation: $T_{\mathcal{M}}(w) := n$ number of steps upon input w.

C_n: stop configuration C_0: start config of $\mathcal{M}$ on input w.

Computation

Notation:

- $C \vdash_{\mathcal{M}} C'$ if $\mathcal{M}$ can change from configuration C to C' in one step.
- $C \vdash^*_{\mathcal{M}} C'$ if $\mathcal{M}$ can change from configuration C to C' in arbitrarily many steps.

The *computation* of a TM $\mathcal{M}$ on input $w \in \Sigma^*$ is either

- an infinite sequence $C_0 \vdash_{\mathcal{M}} C_1 \vdash_{\mathcal{M}} C_2 \ldots$ of configurations or
- a finite sequence $C_0 \vdash_{\mathcal{M}} C_1 \vdash_{\mathcal{M}} C_2 \cdots \vdash_{\mathcal{M}} C_n$ of configurations.

  In the latter case we say that $\mathcal{M}$ *halts* on input w.

  Notation: $T_{\mathcal{M}}(w) := n$ number of steps upon input w.

C_n: stop configuration C_0: start config of $\mathcal{M}$ on input w.

A TM *halts on input w* (and generates output o) if the computation of M on w terminates in configuration

$$(q, (w_1, \ldots, w_{k-1}, o), (p_1, \ldots, p_k)) \quad \text{with} \quad q \in F.$$

Computing a Function and Running Time

Definition:

Let Σ be a finite alphabet.
$f : \Sigma^* \to \Sigma^*$
$g : \mathbb{N} \to \mathbb{N}$
$\mathcal{M}$ be a Turing machine

$\mathcal{M}$ *computes* f *in time* $g(n)$ if for every $w \in \Sigma^*$ $\mathcal{M}$ halts on input w after at most $g(|w|)$ steps with $f(w)$ on its output (last) tape.

(i.e. $T_{\mathcal{M}}(w) \leq g(|w|)$)

Example: The following 2-tape Turing machine

$$\mathcal{M} := \big(\{q_0, q_1, q_f\}, \{a, b\}, \{a, b, \Box\}, \delta, q_0, \{q_f\}\big)$$

where

$$\delta := \left\{ \begin{array}{l} \left(q_0, \binom{a}{-}, \binom{a}{-}, \binom{1}{0}, q_0\right) \\ \left(q_0, \binom{b}{-}, \binom{b}{-}, \binom{1}{0}, q_0\right) \\ \left(q_0, \binom{\Box}{-}, \binom{\Box}{-}, \binom{-1}{0}, q_1\right) \\ \left(q_1, \binom{a}{-}, \binom{\Box}{a}, \binom{-1}{1}, q_1\right) \\ \left(q_1, \binom{b}{-}, \binom{\Box}{b}, \binom{-1}{1}, q_1\right) \\ \left(q_1, \binom{\Box}{-}, \binom{\Box}{-}, \binom{0}{0}, q_f\right) \end{array} \right\}$$

computes the *reverse*-function $reverse(a_1 \ldots a_n) := a_n \ldots a_1$

Example

Example: The following 2-tape Turing machine

$$\mathcal{M} := \left(\{q_0, q_1, q_f\}, \{a, b\}, \{a, b, \Box\}, \delta, q_0, \{q_f\}\right)$$

where

$$\delta := \left\{ \begin{array}{l} \left(q_0, \binom{a}{-}, \binom{a}{-}, \binom{1}{0}, q_0\right) \\ \left(q_0, \binom{b}{-}, \binom{b}{-}, \binom{1}{0}, q_0\right) \\ \left(q_0, \binom{\Box}{-}, \binom{\Box}{-}, \binom{-1}{0}, q_1\right) \\ \left(q_1, \binom{a}{-}, \binom{\Box}{a}, \binom{-1}{1}, q_1\right) \\ \left(q_1, \binom{b}{-}, \binom{\Box}{b}, \binom{-1}{1}, q_1\right) \\ \left(q_1, \binom{\Box}{-}, \binom{\Box}{-}, \binom{0}{0}, q_f\right) \end{array} \right\}$$

computes the *reverse*-function $reverse(a_1 \ldots a_n) := a_n \ldots a_1$ in time $g(n) = 2n + 2 = \mathcal{O}(n)$.

Next: Decision Problems and Turing Acceptors

Travelling Salesman Problem (TSP): Given pairwise distances between cities, you might ask for

- the shortest tour
- the length of the shortest tour

Decision version: given the pairwise distances and a number k, is there a tour of length at most k?

General claim: ability to solve the decision version is "good enough" (why?).

similarly for other problems, e.g. CLIQUE, DOMINATING SET, ...
Decision problems $\longrightarrow$ yes-instances, no-instances.
Next: TMs for decision problems.

Turing Acceptors

Turing machines $\mathcal{M} := \left(Q, \Sigma, \Gamma, \delta, q_0, F\right)$ with set $F \subseteq Q$ is the "final" (or "accepting") states:

- $q \in F$: *accept*

Input $w \in \Sigma^*$ is *accepted* by an acceptor $\mathcal{M}$ if $\mathcal{M}$ halts after finitely many steps in a state $q \in F$.

We say: $\mathcal{M}$ *accepts* input w. Otherwise $\mathcal{M}$ *rejects* the input.

variants: just one accepting state q_a; set of rejecting states F_r

Recall: Inputs come from Σ^*, words over Σ.

Hence: Acceptors accept *languages* $L \subseteq \Sigma^*$

Recall: Languages

Definition/notation

The language $\mathcal{L}(\mathcal{M}) \subseteq \Sigma^*$ *accepted* by a Turing acceptor $\mathcal{M} := \big(Q, \Sigma, \Gamma, \delta, q_0, F\big)$ is defined as

$$\{w \in \Sigma^* : \mathcal{M} \text{ accepts } w\}.$$

(Note that we do not require $\mathcal{M}$ to halt on rejected inputs.)

A language $\mathcal{L} \subseteq \Sigma^*$ is *recursively enumerable*, or *acceptable*, if there is an acceptor $\mathcal{M}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{M})$.

A language $\mathcal{L} \subseteq \Sigma^*$ is *decidable* if there is an acceptor $\mathcal{M}$ such that for all $w \in \Sigma^*$:

$$\begin{array}{lcl} w \in \mathcal{L} & \Longrightarrow & \mathcal{M} \text{ halts on input } w \text{ in an accepting state} \\ w \notin \mathcal{L} & \Longrightarrow & \mathcal{M} \text{ halts on input } w \text{ in a rejecting state} \end{array}$$

Examples

Examples for languages:

- **Regular languages**

  Any regular language has Turing acceptors that can decide its "word problem"

- **The language containing all valid C programs.**

  A Turing acceptor deciding this language is just a syntax checker for C.

- **The language containing all C programs that never run into an infinite loop.**

  A Turing acceptor for this language would be very interesting for software verification.

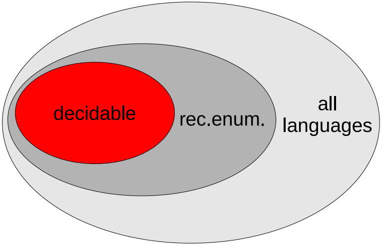
Decidable and Enumerable Languages

Proposition.

1. If a language $\mathcal{L}$ is *decidable* then it is *recursively enumerable*
2. If $\mathcal{L}$ and $\Sigma^* \setminus \mathcal{L}$ are *recursively enumerable* then $\mathcal{L}$ is decidable.

Proposition.

1. If a language $\mathcal{L}$ is *decidable* then it is *recursively enumerable*
2. If $\mathcal{L}$ and $\Sigma^* \setminus \mathcal{L}$ are *recursively enumerable* then $\mathcal{L}$ is decidable.



Note: recursively enumerable a.k.a. *semi-decidable*, *partially decidable*

Decidable and Undecidable Languages

Decidable languages.

- Regular languages
- The language containing all valid C programs

Undecidable languages.

We will see later that the language

- The language containing all C programs that never run into an infinite loop.

is not decidable.

Problems as languages

Executive summary of next few slides

Decision problems can be thought of as language recognition problems, even if, say, the problem involves graphs and we are not explicit about the language.

Encoding of Problems

Languages and Problems:

In many cases, the problems we are interested in are not about words

Instead we are interested in more general structures:

- *graphs*
- *mathematical structures*, e.g. matrices, groups, ...
- *digital circuits*
- ...

Encoding of Problems

Languages and Problems:

In many cases, the problems we are interested in are not about words

Instead we are interested in more general structures:

- *graphs*
- *mathematical structures*, e.g. matrices, groups, ...
- *digital circuits*
- ...

However: Memory of a computer is a linear sequence of *bits*,

i.e. a sequence/word over $\{0, 1\}$.

And so are the input- and work-tapes of Turing machines.

Hence, we need to encode graphs, ... as strings over a finite alphabet.

Encoding of Problems

Encoding schemes:

- To input problems to a computer, each *instance* must be *encoded* as a string of symbols over some alphabet.
- To do this we need an encoding scheme.

Requirement:

Encoding of a problem should not change its essential nature

In particular, it should not essentially change the complexity of a problem

The encoding must be *concise*:

- Represent numerical information efficiently (not in base 1!)
- No unnecessary information (e.g. the solution!), or padding

Languages and Problems

Let $\langle ... \rangle$ be an encoding scheme on graphs and numbers.

Example:

Recall CLIQUE:

Given G, k, does G have a clique of order $\geq k$?

Languages and Problems

Let $\langle ... \rangle$ be an encoding scheme on graphs and numbers.

Example:

Recall CLIQUE:

Given G, k, does G have a clique of order $\geq k$?

Associate CLIQUE with the class

clique $:= \{(G, k) :$ G is a graph containing a clique of order $\geq k\}$.

Languages and Problems

Let $\langle ... \rangle$ be an encoding scheme on graphs and numbers.

Example:

Recall CLIQUE:

Given G, k, does G have a clique of order $\geq k$?

Associate CLIQUE with the class

clique $:= \big\{(G, k) : G$ is a graph containing a clique of order $\geq k\big\}$.

and hence with the language

$$\mathcal{L}\big(Clique\big) := \{\langle G, k\rangle : (G, k) \in \text{clique}\}.$$

Solving CLIQUE is equivalent to deciding $\mathcal{L}\big(Clique\big)$.

Languages and Problems

Let $\langle ... \rangle$ be an encoding scheme on graphs and numbers.

Example:

Recall CLIQUE:

Given G, k, does G have a clique of order $\geq k$?

Associate CLIQUE with the class

clique $:= \big\{(G, k) : G$ is a graph containing a clique of order $\geq k\big\}$.

and hence with the language

$$\mathcal{L}\big(Clique\big) := \{\langle G, k\rangle : (G, k) \in \text{clique}\}.$$

Solving CLIQUE is equivalent to deciding $\mathcal{L}\big(Clique\big)$.

Notation. Let $\mathcal{P}$ be a problem.

We write $\mathcal{L}(\mathcal{P})$ for the language containing string-encodings of *yes*-instances of $\mathcal{P}$.

A Note on Alphabets

Translation between alphabets.

Let $\Sigma := \{a_1, \ldots a_n\}$ be an alphabet; $\mathcal{L} \subseteq \Sigma^*$ a language over Σ.

We can translate $\mathcal{L}$ into $\mathcal{L}' \subseteq \{0,1\}^*$ of the same "complexity".

I.e., encode $a_i \in \Sigma$ as a $\lceil \log |\Sigma| \rceil$-bit binary representation of i and define $\mathcal{L}' := \{\sigma(w_1) \ldots \sigma(w_n) : w_1 \ldots w_n \in \mathcal{L}\}$

Convention.

- assume unless told otherwise that $\Sigma := \{0,1\}$ and $\Gamma := \Sigma \cup \{\Box\}$.
- However, for convenience, we will use different alphabets in concrete examples and constructions.