Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 4: Undecidability

Aalto University
School of Science
Department of Computer Science

Spring 2017

Agenda

- Universal Turing machine
- Halting problem
- Undecidability

(C. Papadimitriou: *Computational Complexity*, Chapters 3.1–3.3)

Some Recap

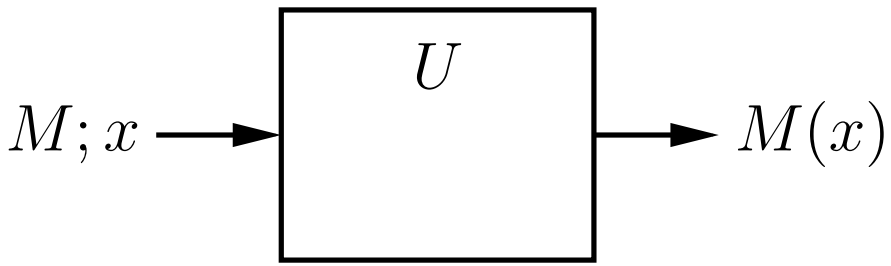
- Let $L \subseteq (\Sigma - \{\sqcup\})^*$ be a language.
- A Turing machine M *decides* L if for every string $x \in (\Sigma - \{\sqcup\})^*$,
  if $x \in L$, then $M(x) =$ "yes" and
  if $x \notin L$, then $M(x) =$ "no" .
- If L is decided by a Turing machine, L is a *recursive* or *decidable* language.
- If L is not recursive, then it is *nonrecursive* or *undecidable*.
- A Turing machine M *accepts* L if for every string $x \in (\Sigma - \{\sqcup\})^*$,
  if $x \in L$, then $M(x) =$ "yes" but if $x \notin L$, then $M(x) = \nearrow$.
- If L is accepted by some Turing machine, L is a *recursively enumerable* or *semidecidable* language.

Proposition

If L is recursive, then it is recursively enumerable.

1. Universal Turing Machine

- Computers are programmable ...
  but a TM has a fixed program which solves a single problem?
- A *universal Turing machine* U
  - takes as input a description of another Turing machine M and an input x for M, and
  - then simulates M on x so that $U(M;x) = M(x)$.
- Compare: a CPU or a virtual machine (JVM etc.)

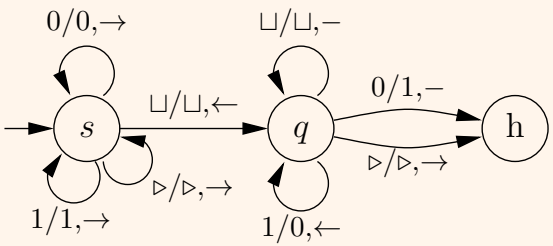


- Note: the symbols M and x are also used to denote the descriptions of M and x.

Encoding TMs using integers

- Encoding a Turing machine $M = (K, \Sigma, \delta, s)$ using integers:
  - $\Sigma = \{1, 2, \ldots, |\Sigma|\}$
  - $K = \{|\Sigma|+1, |\Sigma|+2, \ldots, |\Sigma|+|K|\}$
  - $s = |\Sigma|+1$
  - $|\Sigma|+|K|+1, |\Sigma|+|K|+2, \ldots, |\Sigma|+|K|+6$ encode $\leftarrow, \rightarrow, -, \mathrm{h}$, "yes", "no", respectively.
- An entire TM $M = (K, \Sigma, \delta, s)$ is encoded as $\mathrm{b}(|\Sigma|); \mathrm{b}(|K|); \mathrm{e}(\delta)$ where all integers i are represented as $\mathrm{b}(i)$ with exactly $\lceil \log(|\Sigma|+|K|+6) \rceil$ bits and $\mathrm{e}(\delta)$ is a sequence of pairs $((q, \sigma), (p, \rho, D))$ describing the transition function δ.

Example



with

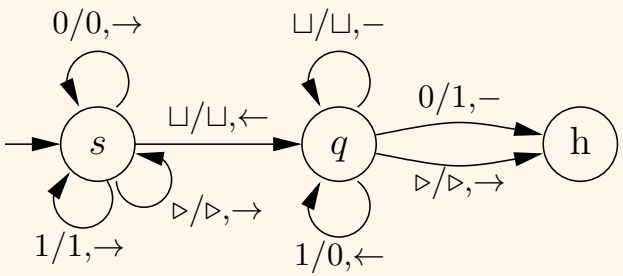
0	1	⊔	▷	s	q	←	→	−	h	"yes"	"no"
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
1	2	3	4	5	6	7	8	9	10	11	12

results in the encoding "4;2;((5,1),(5,1,8))((5,2),(5,2,8))..." i.e. "0100;0010;((0101,0001),(0101,0001,1000))((0101,0010),(0101,0010,1000))..."

The Universal Turing Machine

- U simulates M using
  a string S_1 for the description of M and x, and
  a string S_2 for the current configuration (q, w, u) of M.
  Simulation of a step of M is performed as follows:
  1. Scan S_1 for the block of rules matching the current simulated state, as indicated at the beginning of S_2.
  2. Scan the identified block on S_1 for the rule δ matching the currently scanned simulated symbol, as indicated before the second semicolon on S_2.
  3. Implement the rule. (When M halts, so does U.)

Example



with the input 001 and
the configuration $(q, \triangleright 010, \sqcup)$

Contents of the strings of the simulating machine:

S_1	$\triangleright 0100; 0010; ((0101, 0001), (0101, 0001, 1000))...; 0001, 0001, 0010$
S_2	$\triangleright 0110; 0100, 0001, 0010, 0001; 0011$

2. The Halting Problem

- There are uncountably many languages (= decision problems), but only countably many TMs.
  ☞ There are undecidable languages (= problems).

Definition

HALTING:
INSTANCE: The description of a Turing machine M and its input x.
QUESTION: Does M halt on x?

- The corresponding language is defined as

$$H = \{M; x \mid M(x) \neq \nearrow\}.$$

- HALTING turns out to be an undecidable problem, i.e., there is no Turing machine deciding H.

Properties of HALTING

Proposition

HALTING is recursively enumerable (r.e. for short).

Proof

A slight variant U' of the universal Turing machine U *accepts* H: all halting states of U are forced to be "yes" states. Then the following holds:

- If $M;x \in H$, then $M(x) \neq \nearrow$, $U(M,x) \neq \nearrow$, $U'(M,x) =$ "yes".
- If $M;x \notin H$, then $M(x) = U(M,x) = U'(M,x) = \nearrow$.

Properties of HALTING

Proposition

HALTING is not recursive (i.e. is undecidable).

Proof

- Assume that H is recursive, i.e., some M_H *decides* H.
- Consider the following TM D operating on an input M:
  $D(M)$: **if** $M_H(M;M) =$ "yes" **then** $\nearrow$ **else** "yes".
- Assuming that H is recursive leads to a contradiction because there is no satisfactory result for the computation $D(D)$:
  If $D(D) \neq \nearrow$, then $M_H(D;D) =$ "yes" (since M_H decides H), leading to $D(D) = \nearrow$, a contradiction.
  Similarly, if $D(D) = \nearrow$, then $M_H(D;D) =$ "no" and $D(D) \neq \nearrow$, contradiction again.

☞ H is not recursive.

3. Undecidability

- HALTING spawns a range of other undecidable problems using a *reduction technique*.
- Assume two languages, say B and A.
- A *reduction from B to A* is a transformation t (computable by a Turing machine) of the input y of B to the input $t(y)$ of A such that, for *all* strings y, it holds that

$$y \in B \text{ if and only if } t(y) \in A$$

- In this lecture, we do not impose time or space restrictions on reductions; this will change later.

- To show a problem A undecidable, it suffices to establish that if there is an algorithm (Turing machine) for deciding A, then there is an algorithm (Turing machine) for deciding HALTING.
- This can be shown by devising a reduction t from HALTING to A.
- This implies that A is undecidable as follows.
  Suppose A were decided by a Turing machine M_A.
  Then H would be decided by a machine M_H that on input $M;x$
  - first runs the machine M_t computing the transformation t,
  - then runs M_A on the result.

  In a programming notation:

$$M_H(M;x): \quad y \leftarrow M_t(M;x);\ \textbf{if}\ M_A(y) = \text{“yes”}\ \textbf{then}\ \text{“yes”}\ \textbf{else}\ \text{“no”}.$$

Further undecidable languages

The following languages are not recursive:

(a) $T = \{M \mid M \text{ halts on all inputs}\}$ (corresponds to problem TOTAL)

(b) $\{M;x \mid M(x) = y \text{ for some } y\}$

(c) $\{M;x \mid \text{the computation of } M \text{ on input } x \text{ uses all states of } M\}$

(d) $\{M;x;y \mid M(x) = y\}$

Proof sketch for (a)

A reduction of HALTING to TOTAL:

Given input $M;x$, consider a machine M^x that works as follows:
$M^x(y)$: **if** $y = x$ **then** $M(x)$ **else halt**.

Define reduction mapping $t(M;x) = M^x$. (I.e. the input x is hard-coded into the machine code of M and the result is the new code.)

Now $M;x \in H$ iff M halts on x iff M^x halts on all inputs iff $M^x \in T$.

A Property of HALTING

Proposition

The language H is complete for r.e. languages, i.e. any r.e. language L can be reduced to it.

Proof

Let L be any r.e. language, accepted by a TM M_L.

Then L can be reduced to H by the reduction mapping $t(x) = M_L; x$.

This holds as $x \in L$ iff $M_L(x) =$ "yes" iff $M_L(x) \neq \nearrow$ iff $M_L; x \in H$.

Properties of recursive languages

Proposition

If L is recursive, then so is $\overline{L}$ (the complement of L).

Proposition

A language L is recursive iff both L and $\overline{L}$ are recursively enumerable.

Proof sketch

($\Rightarrow$) By the previous proposition and the fact that every recursive language is also recursively enumerable.

($\Leftarrow$) Simulate M_L and $M_{\overline{L}}$ on input x in an interleaved fashion:

– If M_L accepts, return "yes" and

– if $M_{\overline{L}}$ accepts, return "no".

☞ The complement $\overline{H}$ of H *is not recursively enumerable*.

Recursively enumerable languages

Proposition

A language L is recursively enumerable iff there is a machine M such that $L = E(M) = \{x \mid (s, \triangleright, \varepsilon) \overset{M}{\to}{}^{*} (q, y \sqcup x \sqcup, \varepsilon)\}$.

Any non-trivial property of Turing machines is undecidable:

Theorem (Rice's Theorem)

Let C be a proper non-empty subset of r.e. languages. Then the following problem is undecidable: given a Turing machine M, is $L(M) \in C$?

Here $L(M)$ is the language accepted by a Turing machine M.

Learning Objectives

- The definitions of recursive and recursively enumerable languages (including examples of such languages).
- Ability to argue that a given problem is undecidable using the reduction technique.