T-79.5103 Computational Complexity Theory

Lecture 4: Undecidability

Aalto University
School of Science
Department of Computer Science

Spring 2016
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 $M(x) = \text{"yes"}$ and
  if $x \notin L$, then $M(x) = \text{"no"}$.
- 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 $M(x) = \text{"yes"}$ but if $x \notin L$, then $M(x) = \uparrow$.
- 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 $\leftarrow, \rightarrow, -, h, \text{“yes”}, \text{“no”}$, respectively.

- An entire TM $M = (K, \Sigma, \delta, s)$ is encoded as $b(|\Sigma|); b(|K|); e(\delta)$ where all integers $i$ are represented as $b(i)$ with exactly $\lceil \log(|\Sigma| + |K| + 6) \rceil$ bits and $e(\delta)$ is a sequence of pairs $((q, \sigma), (p, \rho, D))$ describing the transition function $\delta$.

**Example**

```
0/0, →
□/□, ←
↓/↓, →
1/1, →
```

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 \( \delta \) 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 \((q, \triangleright 010, \square)\)

Contents of the strings of the simulating machine:

\[
S_1 \begin{array}{c}
\triangleright 0100; 0010; ((0101, 0001), (0101, 0001, 1000))\ldots; 0001, 0001, 0010
\end{array}
\]

\[
S_2 \begin{array}{c}
\triangleright 0110; 0100, 0001, 0010, 0001; 0011
\end{array}
\]
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
  \[ H = \{ M; x \mid M(x) \neq \rangle \}. \]
- 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 $M(x) \not\xrightarrow{\text{halt}}$, $U(M, x) \not\xrightarrow{\text{halt}}$, $U'(M, x) = \text{"yes"}$.
- If $M; x \not\in H$, then $M(x) = U(M, x) = U'(M, x) = \xrightarrow{\text{halt}}$. 
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 $M$:
  
  
  if $M_H(M; M) = \text{"yes"}$ then $\uparrow$ else $\text{"yes"}$.

- Assuming that $H$ is recursive leads to a contradiction because there is no satisfactory result for the computation $D(D)$:
  
  If $D(D) \neq \uparrow$, then $M_H(D; D) = \text{"yes"}$ (since $M_H$ decides $H$), leading to $D(D) = \uparrow$, a contradiction.
  
  Similarly, if $D(D) = \uparrow$, then $M_H(D, D) = \text{"no"}$ and $D(D) \neq \uparrow$, contradiction again.

Therefore, $H$ 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:

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

\( M^x(y) : \text{ if } y = x \text{ then } M(x) \text{ else } \text{halt}. \)

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 $M_L(x) \neq \uparrow$ 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 \( L = E(M) = \{ x \mid (s, \triangleright, \varepsilon) \xrightarrow{M}^* (q, y \sqcup x \sqcup, \varepsilon) \} \).

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.