Computational Models — Lecture 9[1]

Handout Mode

Iftach Haitner.

Tel Aviv University.

December 26, 2016

[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Talk Outline

- Computable functions
- Reducibility
- Mapping reductions
- Busy Beaver

- Sipser's book, 5.1 – –5.2

Reminder

We have already

- Established Turing Machines as the gold standard of computers and computability . . .
- Seen examples of decidable problems . . .
- Saw that

$$A_{TM} = \{\langle M, w\rangle : M \text{ is a TM that accepts } w\}$$

and

$$H_{TM} = \{\langle M, w\rangle : M \text{ is a TM and } M \text{ halts on input } w\}$$

are undecidable.

Today, we look at other computationally undecidable problems via reductions and introduce the techniques of mapping reductions.

Section 1

Computable Functions

Computable functions

Definition 1 (total computable functions)

A TM M computes a total function $f : \Sigma^* \mapsto \Sigma^*$, if when starting with an input w, M halts with (only) $f(w)$ written on tape.[a]

[a] The definition naturally extends to functions of more than one variable, where some special separator symbol indicates end of one variable and beginning of next.

Computable functions are also called recursive functions.

Example, 1

Claim 2

All the "usual" arithmetic functions on integers are computable.

These include addition, subtraction, multiplication, division (quotient and remainder), exponentiation, roots (to a specified precision), modular exponentiation, greatest common divisor.

Even non-arithmetic functions, like logarithms and trigonometric functions, can be computed (to a specified precision), using Taylor expansion or other numeric mathematic techniques.

Exercise 3

Design a TM that on input $\langle m, n \rangle$, halts with $\langle m + n \rangle$ on tape.

Example, 2

A useful class of functions modifies TM descriptions. For example:

Algorithm 4

On input w:
If $w = \langle M \rangle$ for some TM M, return $\langle \widetilde{M} \rangle$, where $\widetilde{M}$ is TM s.t.,

- $L(\widetilde{M}) = L(M)$, and
- $\widetilde{M}$ does not halt on $x \notin L(\widetilde{M})$

Otherwise, return ε.

Question 5

Is the function defined above total? computable?

Are all total functions computable?

No, $f(\langle M, w \rangle)$ that returns 1 if M accepts w, and 0 otherwise, is not computable.

Definition of $\widetilde{M}$

Given $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, TM $\widetilde{M} = (Q', \Sigma, \Gamma, \delta', q_0, q_a, q_r)$ is defined by

- $Q' = Q \cup \{q^*\}$
-

$$\delta'(q,\sigma) := \begin{cases} (q^*, \sigma, R), & \delta(q,\sigma) = (q_r, \cdot, \cdot) \\ (q^*, \sigma, R) & q = q^* \\ (q', \sigma', D), & \delta(q,\sigma) = (q', \sigma', D) \wedge q \notin \{q_r, q^*\} \end{cases}$$

Section 2

Reducibility

Reducibility

- Finding your way around a new city, reduces to . . .
  obtaining a city map.
- Finding the median in an array, reduces to . . . sorting an array
- The core idea behind procedures

Reducibility, in our context

Involves two languages A and B.
Desired property: If A reduces to B, then any solution of B can be used to find a solution of A.

This property says nothing about solving A by itself, or B by itself.

but if A reduces to B, then A cannot be harder than B:

- if B is decidable, so is A.
- if A is undecidable, then B is undecidable.

We next use reductions and the undecidability of A_{TM}, to show the undecidability of several problems.

Reminder, H_{TM} is undecidable

- $A_{TM} = \{\langle M, w\rangle : M \text{ is a TM that accepts } w\}$
- $H_{TM} = \{\langle M, w\rangle : M \text{ is a TM and } M \text{ halts on input } w\}$

Theorem 6

H_{TM} *is undecidable.*

Proof: Assume, by way of contradiction, that $\exists$ TM R that decides H_{TM}.

Algorithm 7 (S)

On input $\langle M, w\rangle$.
Reject, if input is not well formatted.

1. Emulate R on $\langle M, w\rangle$.
2. If R rejects, reject.
3. If R accepts, emulate M on w until it halts.
4. Accept if M accepted; otherwise reject.

TM S decides A_{TM}, a contradiction ♣
What we actually did is a reduction from A_{TM} to H_{TM}.

EMPTY_{TM} is undecidable

$$\text{EMPTY}_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM and } L(M) = \emptyset\}$$

Theorem 8

EMPTY_{TM} *is undecidable.*

Proof's idea: By contradiction:

- Assume EMPTY_{TM} is decidable and let R be a TM that decides EMPTY_{TM}.
- Use R to construct S, a TM that decides A_{TM}.

Algorithm 9 (S – first attempt)

On input $\langle M, w \rangle$:
Emulate $R(\langle M \rangle)$ and reject if R accepts.

But what if R rejects?

Solution? Modify M.

EMPTY_{TM} is undecidable, the TM M_w

Definition 10 (M_w)

For TM M and input w, the TM M_w is defined as follows:

On input x, emulate M on w, and accept if M does.

$$\text{L}(M_w) = \begin{cases} \Sigma^* & w \in \text{L}(M) \\ \emptyset & \text{otherwise} \end{cases}$$

Question 11

Can a TM construct M_w from $\langle M, w \rangle$?

Answer: Easily because we need only hardwire w, and add a few extra states to replace x with w.

EMPTY_{TM} is undecidable, the reduction

- $\text{EMPTY}_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM and } \text{L}(M) = \emptyset\}$
- Assume EMPTY_{TM} is decidable and let R be a TM that decides EMPTY_{TM}.

Algorithm 12 (S)

On input $\langle M, w \rangle$. Reject, if input is not well formatted.

1. Construct M_w.
2. Emulate R on input $\langle M_w \rangle$.
3. Accept if R Rejects; reject if R accepts.

Claim 13

S decides A_{TM}.

Proof: Recall that $\text{L}(M_w) = \emptyset$ iff M does not accept w.□

A contradiction. ♣

EMPTY_{TM} is undecidable, building M_w

Given $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ and $w = w_1, \ldots w_n$ (assuming $n \geq 1$),

build $M_w = (Q', \Sigma, \Gamma, \delta', \overrightarrow{q}_1, q_a, q_r)$, for
$Q' = Q \cup \{\overrightarrow{q}_1, \ldots \overrightarrow{q}_{n+1}, \overleftarrow{q}, \overleftarrow{q}_1\}$, and δ' defined by

$$\delta'(q,\sigma) := \begin{cases} (\overrightarrow{q}_2, \$, R) & q = \overrightarrow{q}_1 \\ (\overrightarrow{q}_{i+1}, w_i, R) & q = \overrightarrow{q}_i, 2 \leq i \leq n \\ (\overrightarrow{q}_{n+1}, \sqcup, R) & q = \overrightarrow{q}_{n+1}, \sigma \neq \sqcup \\ (\overleftarrow{q}, \sigma, L) & q = \overrightarrow{q}_{n+1}, \sigma = \sqcup \\ (\overleftarrow{q}, \sigma, L) & q = \overleftarrow{q}, \sigma \neq \$ \\ (\overleftarrow{q}_1, w_1, R) & q = \overleftarrow{q}, \sigma = \$ \\ (q_0, \sigma, L) & q = \overleftarrow{q}_1 \\ (q', \sigma', D), & \delta(q,\sigma) = (q', \sigma', D) \\ & \text{and none of the above conditions hold} \end{cases}$$

REG_{TM} is undecidable

$$\text{REG}_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM and } \text{L}(M) \text{ is regular}\}$$

Theorem 14

REG_{TM} *is undecidable.*

Proof's idea: By contradiction.

- Assume REG_{TM} is decidable and let R be a TM that decides REG_{TM}.
- Use R to construct S – a TM that decides A_{TM}.

Question 15

But how we construct S?

Intuition: On input $\langle M, w \rangle$, build M_w that accepts regular language iff M accepts w.

REG_{TM} is undecidable, the TM M_w

The TM M_w will have the following property.

- if M does not accept w, then $\text{L}(M_w) = \{0^n 1^n : n \geq 0\}$
- if M does accepts w, then $\text{L}(M_w) = \Sigma^*$

Algorithm 16 (M_w)

On input x,

1. Accept, if x is of the form $0^n 1^n$.
2. (otherwise) Emulate M on input w and accept if M does.

Claim 17

1. If M does not accept w, then $\text{L}(M_w)$ is not regular.
2. If M does accept w, then $\text{L}(M_w)$ is regular.
3. The function $f(\langle M, w \rangle) = \langle M_w \rangle$ is computable.

REG_{TM} is undecidable, the reduction

Algorithm 18 (S)

On input $\langle M, w \rangle$. Reject, if input is not well formatted.

1. Construct M_w.
2. Emulate R on input $\langle M_w \rangle$ (recall that R is a TM that decides REG_{TM}).
3. Accept, if R accepts ; Reject, if R rejects.

Claim 19

S decides A_{TM}.

A contradiction ♣

EQ_{TM} is undecidable

$$EQ_{TM} = \{\langle M_1, M_2\rangle : M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Theorem 20

EQ_{TM} *is undecidable.*

We are getting tired of reducing A_{TM} to everything.
Let's try instead a reduction from $EMPTY_{TM}$ to EQ_{TM}.

Proof's idea:

- $EMPTY_{TM}$ is the problem of testing whether a TM language is empty.
- EQ_{TM} is the problem of testing whether two TM languages are the same.
- If one of these two TM languages happens to be empty, then we are back to $EMPTY_{TM}$.
- So $EMPTY_{TM}$ is a special case of EQ_{TM}.

The rest is easy.

EQ_{TM} is undecidable, 2

- $EQ_{TM} = \{\langle M_1, M_2\rangle : M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$
- Assume EQ_{TM} is decidable and let R be a TM deciding EQ_{TM}.

Proof:

Algorithm 21 (M_{NO})

On input x, reject

Algorithm 22 (S)

On input $\langle M\rangle$. Reject, if input is not well formatted.
Emulate R on input $\langle M, M_{NO}\rangle$.

If R accepts, accept; if R rejects, reject.

Claim 23

S decides $EMPTY_{TM}$.

A contradiction ♣

Bucket of undecidable problems

Similar techniques can be used to prove undecidability of

- Does a TM accept a decidable language?
- Does a TM accept a context-free language?
- Does a TM accept a finite language?
- Does a TM halt on all inputs?
- Is there an input string that causes a TM to traverse all its states?

Section 3

Mapping Reductions

Motivation

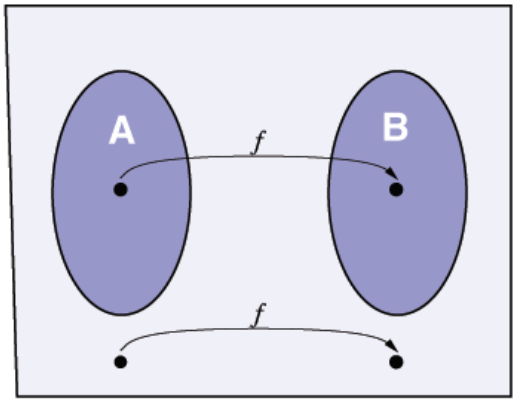
So far, we have seen many examples of reductions from one language to another, but the notion was neither defined nor treated formally.

Reductions play an important role in

- decidability theory (here and now)
- complexity theory (to come)

Time to get formal.

Mapping Reductions



Definition 24

A total computable function $f : \Sigma^* \mapsto \Sigma^*$ is a mapping reduction from language A to language B, if $x \in A \iff f(x) \in B$, for every $x \in \Sigma^*$.

If a mapping reduction from A to B exists, we say that A is mapping reducible to B, denoted by $A \leq_m B$.

A mapping reduction converts questions about membership in A to membership in B.

Remark 25

Note that $A \leq_m B \iff \overline{A} \leq_m \overline{B}$

Applications

Theorem 26

If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: Let M be the decider for B, and f a (mapping) reduction from A to B.

Algorithm 27 (N)

On input x

1. Compute $f(x)$
2. Emulate M on input $f(x)$. Accepts iff M does.

♣

Corollary 28

If $A \leq_m B$ and A is undecidable, then B is undecidable.

In fact, this has been our principal tool for proving undecidability of languages other than A_{TM}

H_{TM} is undecidable — Revisited

- $A_{TM} = \{\langle M, w\rangle : M \text{ is a TM that accepts } w\}$
- $H_{TM} = \{\langle M, w\rangle : M \text{ is a TM and } M \text{ halts on input } w\}$

Claim 29

$A_{TM} \leq_m H_{TM}$

The following computable function f establishes $\langle M, w\rangle \in A_{TM} \iff f(\langle M, w\rangle) \in H_{TM}$.

Definition 30 (f)

On input $x = \langle M, w\rangle$. If x is not well formatted, return (some fixed) $x' \notin H_{TM}$.
Otherwise, return $\langle \widetilde{M}, w\rangle$.

Definition 31 (TM $\widetilde{M}$)

On input y: emulate $M(y)$. Accept, if M accepts; enter a loop, if M rejects.

$H_{TM,\varepsilon}$ is undecidable

Recall that

- $H_{TM} = \{\langle M, w\rangle : M \text{ is a TM that halts on input } w\}$
- $H_{TM,\varepsilon} = \{\langle M\rangle : M \text{ is a TM that halts on input } \epsilon\}$

Claim 32

$H_{TM} \leq_m H_{TM,\varepsilon}$

The following computable function f establishes $\langle M, w\rangle \in H_{TM} \iff f(\langle M, w\rangle) \in H_{TM,\varepsilon}$.

Definition 33 (f)

On input $x = \langle M, w\rangle$. If x is not well formatted, return $x' \notin H_{TM,\varepsilon}$. Otherwise, return $\langle M_w\rangle$.

(Recall M_w: on input x, emulates M on w).

Note that M_w halts on $\varepsilon \iff M$ halts on w.

Therefore, $\langle M\rangle \in H_{TM,\varepsilon} \iff \langle M, w\rangle \in H_{TM,\varepsilon}$.

Mapping reducible relation is not symmetric

Claim 34

Let $\mathcal{A} \in \mathcal{R}$ and $\mathcal{B} \notin \mathcal{R}$, then $\mathcal{A} \leq_m \mathcal{B}$, but $\mathcal{B} \not\leq_m \mathcal{A}$.

Proof: It is clear that $\mathcal{B} \not\leq_m \mathcal{A}$ (?)

For proving $\mathcal{A} \leq_m \mathcal{B}$, define f as follows:

Definition 35 (f)

On input w, return $x_Y \in \mathcal{B}$ if $w \in \mathcal{A}$, and $x_N \notin \mathcal{B}$ otherwise.

- Do x_Y and X_N (always) exist?
- How can the TM implementing f find them?

♣

Enumerability

Theorem 36

If $A \leq_m B$ and $B \in \mathcal{RE}$, then $A \in \mathcal{RE}$.

Proof is same as in the decidability case, using accepters instead of deciders.

Corollary 37

If $A \leq_m B$ and $A \notin \mathcal{RE}$, then $B \notin \mathcal{RE}$.

TM equality, revisited

$$\mathrm{EQ}_{\mathrm{TM}} = \{\langle M_1, M_2\rangle : M_1, M_2 \text{ are TMs and } \mathrm{L}(M_1) = \mathrm{L}(M_2)\}$$

Theorem 38

Both $\mathrm{EQ}_{\mathrm{TM}}$ *and, its complement,* $\overline{\mathrm{EQ}_{\mathrm{TM}}}$*, are not enumerable.*

Stated differently, $\mathrm{EQ}_{\mathrm{TM}}$ is neither enumerable nor co-enumerable, or $\mathrm{EQ}_{\mathrm{TM}} \notin \mathcal{RE} \cup co\text{-}\mathcal{RE}$.

- We show (next slide) that $\mathrm{A}_{\mathrm{TM}} \leq_m \mathrm{EQ}_{\mathrm{TM}}$, and $\mathrm{A}_{\mathrm{TM}} \leq_m \overline{\mathrm{EQ}_{\mathrm{TM}}}$.
- Hence, $\overline{\mathrm{A}_{\mathrm{TM}}} \leq_m \overline{\mathrm{EQ}_{\mathrm{TM}}}$ and $\overline{\mathrm{A}_{\mathrm{TM}}} \leq_m \mathrm{EQ}_{\mathrm{TM}}$ (?)
- Hence, neither $\mathrm{EQ}_{\mathrm{TM}}$ nor $\overline{\mathrm{EQ}_{\mathrm{TM}}}$ are enumerable.

$A_{TM} \leq_m EQ_{TM}$

Claim 39

$A_{TM} \leq_m EQ_{TM}$.

Definition 40 (f)

On input $x = \langle M, w \rangle$. If x is not well formatted, return $x' \notin EQ_{TM}$.

1. Construct the TM M_w (recall that M_w emulates M on w [while ignoring its input]), and the TM M_{all} that accepts Σ^*.
2. Return $\langle M_w, M_{all} \rangle$.

- If M accepts w, then M_w accepts everything. Otherwise, M_w accepts nothing.
- Hence, $\langle M, w \rangle \in A_{TM} \iff \langle M_w, M_{all} \rangle \in EQ_{TM}$.

$A_{TM} \leq_m \overline{EQ_{TM}}$

Claim 41

$A_{TM} \leq_m \overline{EQ_{TM}}$.

Definition 42 (f)

On input $x = \langle M, w \rangle$. If x is not well formatted, return $x' \notin \overline{EQ_{TM}}$.

1. Construct the TM M_w, and the TM M_{NO} that accepts $\emptyset$.
2. Return $\langle M_w, M_{NO} \rangle$.

- If M accepts w, then M_w accepts everything. Otherwise, M_w accepts nothing.
- Hence, $\langle M, w \rangle \in A_{TM} \iff \langle M_w, M_{NO} \rangle \in \overline{EQ_{TM}}$

Section 4

Busy Beaver

The *Busy Beaver*

(taken from `http://www.saltine.org/joebeaver1.jpg`)

The *Busy Beaver*

We focus on one tape TMs, with $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \sqcup\}$.

Definition 43 (S_n and $BB(n)$)

For $n \in \mathbb{N}$, let

- $S_n = \{$ all n-state TM's that halt on $\varepsilon\}$.
- $\text{BB}(n) = \max_{M \in S_n}\{$ # of steps taken by M on input $\varepsilon\}$.

- The set S_n is finite (under standard encoding)
- Every $M \in S_n$ runs for finitely many steps on ε .
- $\text{BB}(n)$ is a total function from $\mathbb{N}$ to $\mathbb{N}$ (in particular, $\text{BB}(n) \in \mathbb{N}$ for every $n \in \mathbb{N}$).

Values of BB (size not including accept and reject states):

size	2	3	4	5	6
BB	6	21	107	$\geq 47,176,870$	$\geq 7.4 \times 10^{36534}$

The *Busy Beaver* function is not computable

Theorem 44

BB *is not computable.*

Proof: Assume BB is computable by the TM R and consider the undecidable language $\mathrm{H}_{\mathrm{TM},\varepsilon} = \{\langle M \rangle : M \text{ is a TM that halts on } \varepsilon\}$.

Algorithm 45 (S)

On input $\langle M \rangle$

1. Let m be $\#$ of states in M, and let $n = R(m)$.
2. Emulate M on ε for $n+1$ steps.
3. Accept if M halts; otherwise reject

Note that if M did not halt in $n+1$ steps, then it will never halt!

Claim 46

S decides $\mathrm{H}_{\mathrm{TM},\varepsilon}$.

The bounded *Busy Beaver* function is computable

Definition 47

For $d \in \mathbb{N}$, define the function $\mathrm{BB}_d : \mathbb{N} \mapsto \mathbb{N}$ as

$$\mathrm{BB}_d(n) := \begin{cases} \mathrm{BB}(n), & n \leq d, \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 48

The function BB_d is computable for every $d \in \mathbb{N}$.

Proof's idea: "Hardwire" the values $\mathrm{BB}(1), \ldots, \mathrm{BB}(d)$ into a TM to compute BB_d.