REDUCTIONS AND UNDECIDABILITY

Abhijit Das

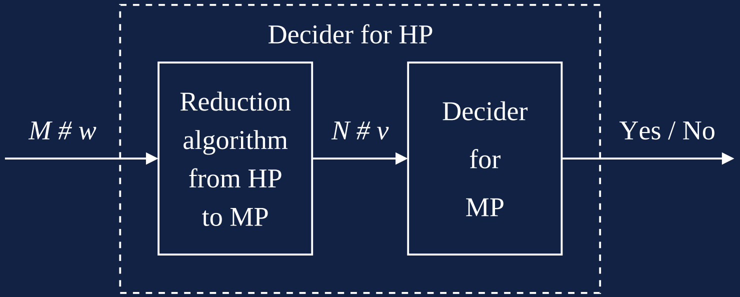
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

March 14, 2021

Diagonalization

- Any Turing machine M can be encoded as a string over $\{0,1\}$.
- Any input w for M can also be encoded as a binary string.
- Two important problems (languages)
  - $\mathrm{MP} = \{M \# w \mid M \text{ accepts input } w\}$.
  - $\mathrm{HP} = \{M \# w \mid M \text{ halts on input } w\}$.
- A total TM (or decider) halts on all inputs.
- Both these problems are Turing-recognizable (r.e.).
- By a diagonalization argument, we have proved HP to be non-recursive.
- No decider can exist for HP, no matter how intelligent Turing machines are.
- A similar diagonalization argument can be made for MP.

Reduction



- We want to prove the undecidability of the MP.
- A reduction algorithm converts an input $M \# w$ for HP to an input $N \# v$ for MP.
- The reduction algorithm is a total Turing machine (halts after each conversion).
- N accepts v if and only if M halts on w.
- If MP has a decider D, then the reduction algorithm followed by D decides HP.
- Contradiction. So a decider of MP cannot exist.

The Reduction Algorithm

Input: M and w.
Output: N and v.
Steps:

- Add a new accept state t' and a new reject state r' to M.
- Mark the old accept and reject states t and r of M as non-halting.
- Add transitions $\delta(t,*) = (t',*,R)$ and $\delta(r,*) = (t',*,R)$.
- Take $v = w$.
- Convince yourself that a total TM can transform (M,w) to (N,v).
- N always rejects by looping (no transition to r' added).
- If M halts after accepting (in state t) or rejecting (in state r), N runs one more step to jump to t' and accepts.
- If M loops on w, N also loops.
- M halts on $w \iff N$ accepts v.

Direction of Reduction

From a problem already known to be undecidable
to a problem which we want to prove to be undecidable.

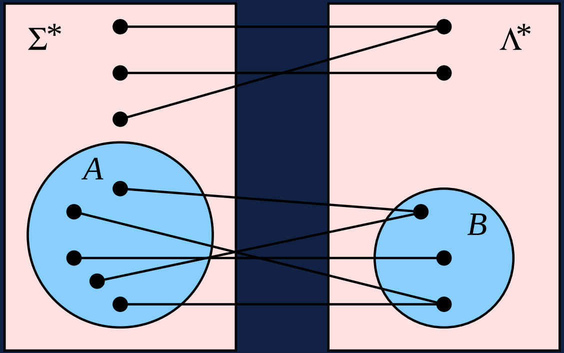
A valid reduction from MP to HP

Input: $M \# w$ for the membership problem
Output: $N \# v$ for the halting problem

- Keep the accept state t of M the same in N.
- Create a new reject state r' for N, and transitions $\delta(r,*) = (r,*,R)$ (loop in state r).
- Take $v = w$.
- M accepts $w \iff N$ halts on v (no transition lets N enter r').

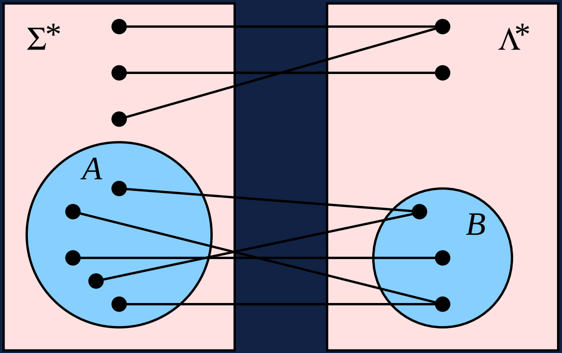
- This is not an undecidability proof for MP. A decider for MP may not be forced to use a (hypothetical) decider for HP.
- If MP was proved to be undecidable, this reduction proves the undecidability of HP.

Formal Definition of Reduction



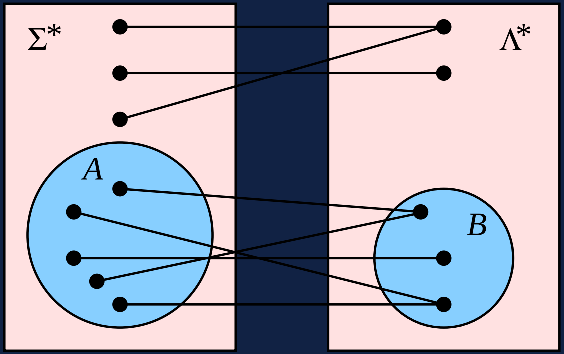
- Let $A \subseteq \Sigma^*$ and $B \subseteq \Lambda^*$ be languages.
- Consider a map $\sigma : \Sigma^* \to \Lambda^*$.
- If $w \in A$, then $\sigma(w) \in B$.
- If $w \in \Sigma^* \setminus A$, then $\sigma(w) \in \Lambda^* \setminus B$.

Formal Definition of Reduction



- σ need not be injective.
- A Turing machine R implements σ.
- On every input w, the TM R halts after correctly computing $\sigma(w)$.
- We call R a reduction algorithm.

Formal Definition of Reduction



- σ is a reduction from A to B.
- Notation: $A \leqslant_m B$ (many-to-one reduction).
- The membership problem for A is no more difficult than the membership problem for B.
- Example: HP $\leqslant_m$ MP and MP $\leqslant_m$ HP.

Notes on Reduction

- A language L can be rephrased as the membership problem:

  $\boxed{\text{Given } w \in \Sigma^*, \text{ is } w \in L?}$

- We talk about reduction of one problem to another.
- For problems P, Q, we can write $P \leqslant_m Q$.
- A reduction algorithm is supposed to convert an instance of P to an instance of Q.
- A reduction algorithm makes no effort to solve either P or Q.
- Two uses of reduction $P \leqslant_m Q$:
  - Given a solver for Q, use this solver as a subroutine to solve P.

    This is one way of solving P, not the only or the most efficient way.
  - If no solver for P exists, then no solver for Q can exist.

Reduction Example 1

Proposition: The problem whether a given Turing machine M accepts the null string ε is undecidable.

Proof Use reduction *from* HP.

$$M \# w \longmapsto N$$

Reduction Example 1

- **Input:** M and w (an instance of HP).
- **Output:** A Turing machine N that accepts ε if and only if M halts on w.
- N can use M and w in any manner it likes.
  - These may be embedded by the reduction algorithm in the finite control of N.
  - Alternatively, the reduction algorithm may copy these to some tapes/tracks of N.
- Behavior of N on input v:
  - Erase input v.
  - Write the string w on the tape.
  - Simulate M on w.
  - If the simulation halts, accept v.
- N accepts its input $v \iff M$ halts on w.
- $\mathscr{L}(N) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } w, \\ \emptyset & \text{if } M \text{ does not halt on } w. \end{cases}$
- In particular, N accepts $\varepsilon \iff M$ halts on w.

Reduction Example 1

The same proof can be used to prove that the following problems are also undecidable.

Proposition: Let w be a fixed string over Σ. The problem whether a given Turing machine M accepts w is undecidable.

Proposition: The problem whether a given Turing machine M accepts any string at all is undecidable.

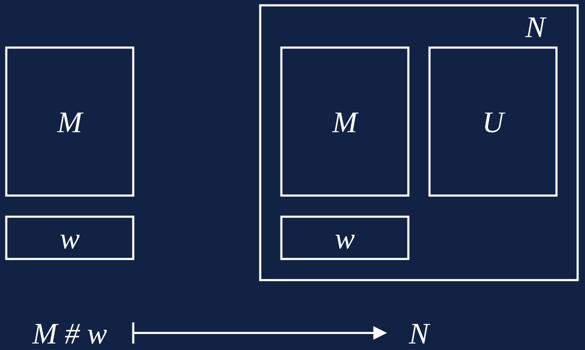
Proposition: The problem whether a given Turing machine M accepts all the strings over Σ is undecidable.

Proposition: The problem whether a given Turing machine M accepts only finitely many strings is undecidable.

Reduction Example 2

Proposition: The problem whether the language of a given Turing machine M is regular is undecidable.

Proof Again use reduction from HP.



Reduction Example 2

- **Input:** An instance for HP (M and w)
- **Output:** A Turing machine N whose language is regular if and only if M halts on w.
- N has the information of M and w embedded in its finite control.
- N embeds the information of another **fixed** Turing machine U in its finite control.
- Take any language L that is recursively enumerable but not recursive.
- Take any TM U whose language is L.
- For example, if $L = \mathrm{MP}$, then U is the Universal Turing Machine.

Reduction Example 2

N, upon the input of v, does the following.

- Store v on a separate tape/track.
- Write w on the tape, and simulate M on w.
- If the simulation halts, do:
  - Simulate U on v.
  - If U accepts v, accept v.

- N accepts v if and only if both the following conditions hold.
  - M halts on w.
  - U accepts (and halts) on v.
- $\mathscr{L}(N) = \begin{cases} L & \text{if } M \text{ halts on } w, \\ \emptyset & \text{if } M \text{ does not halt on } w. \end{cases}$
- $\emptyset$ is regular, but A is not regular.

Reduction Example 2

- Let $L_2 = \{N \mid \mathscr{L}(N) \text{ is regular}\}$.
- We have a reduction from HP to the complement $\overline{L_2}$.
- This proves that $\overline{L_2}$ is not recursive.
- But recursive languages are closed under complementation, so L_2 is not recursive too.
- Alternative argument:
  - Let $\overline{L_2}$ have a decider $\overline{D}$.
  - Then L_2 has a decider D that simulates $\overline{D}$ and flips the decision of $\overline{D}$.
  - The above reduction followed by D decides HP.

Reduction Example 2

The same reduction can be used to prove the following undecidability results.

Proposition: The problem whether the language of a given Turing machine M is finite is undecidable.

Proposition: The problem whether the language of a given Turing machine M is context-free is undecidable.

Proposition: The problem whether the language of a given Turing machine M is context-sensitive is undecidable.

Proposition: The problem whether the language of a given Turing machine M is recursive is undecidable.

Note: The problem whether the language of a given Turing machine M is recursively enumerable is trivially decidable.

A Theorem about Reduction

Theorem: Let A, B be languages along with a reduction $A \leqslant_m B$.
If B is r.e., then A is also r.e.
Contrapositively, if A is not r.e., then B is also not r.e.

Proof

- Let σ be the reduction map from A to B.
- Let $B = \mathscr{L}(N)$ for a Turing machine N.
- A recognizer M for A can be designed as follows.
- On an input w, M does the following:
    - Compute $\sigma(w)$ from w.
    - Run N on $\sigma(w)$.
    - Accept if and only if N accepts $\sigma(w)$.

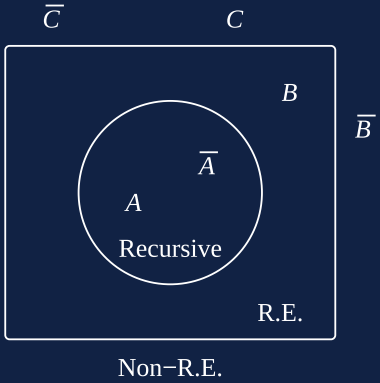
Another Theorem about Reduction

Theorem: Let A, B be languages along with a reduction $A \leqslant_m B$.
If B is recursive, then A is also recursive.
Contrapositively, if A is not recursive, then B is also not recursive.

Proof

- Let B be recursive.
- Let σ be the reduction map $A \leqslant_m B$.
- Since B is r.e., A is r.e. too (by the previous theorem).
- σ is also a reduction map for $\overline{A} \leqslant_m \overline{B}$.
- $\overline{B}$ is recursive and so r.e.
- By the previous theorem, $\overline{A}$ is r.e. too.
- Since A and $\overline{A}$ are both r.e., A is recursive.

Three Possibilities



- If A and $\overline{A}$ are r.e., then both are recursive.
- If B is r.e. but not recursive, then $\overline{B}$ must be non-r.e. Examples: $\overline{\text{HP}}$, $\overline{\text{MP}}$ are non-r.e.
- Both C and $\overline{C}$ can be non-r.e.

An Example of the Third Type

Proposition: Neither the language

$$\mathrm{FIN} = \{M \mid \mathscr{L}(M) \text{ is finite}\}$$

nor its complement $\overline{\mathrm{FIN}}$ is r.e.

- We have proved that FIN is not recursive by reduction from HP.
- This proof cannot establish that FIN is non-r.e.
- We need reduction from a non-r.e. language.
- $\overline{\mathrm{HP}} = \{M \# w \mid M \text{ does not halt on } w\}$ is non-r.e.
- We now show

  $$\overline{\mathrm{HP}} \leqslant_m \mathrm{FIN}$$

  and

  $$\overline{\mathrm{HP}} \leqslant_m \overline{\mathrm{FIN}}.$$

$\overline{\text{HP}} \leqslant_m$ FIN

Input: A TM M and an input w for M.
Output: A TM N such that $\mathscr{L}(N)$ is finite if and only if M does not halt on w.
Note: N has the information of M and w in its finite control.

Behavior of N on input v

- Erase the input v.
- Write w on the tape, and simulate M on w.
- If the simulation halts, accept v.

- If M does not halt on w, $\mathscr{L}(N) = \emptyset$ which is finite.
- If M halts on w, $\mathscr{L}(N) = \Sigma^*$ which is infinite.

Note: The reduction algorithm is not supposed to run N. It only creates a description of N.

$\overline{\text{HP}} \leqslant_m \overline{\text{FIN}}$

Input: A TM M and an input w for M.
Output: A TM N such that $\mathscr{L}(N)$ is infinite if and only if M does not halt on w.
Note: N has the information of M and w in its finite control.

Behavior of N on input v

- Store v on a separate tape/track.
- Write w on the tape, and simulate M on w for at most $|v|$ steps.
- Accept if the simulation does **not** halt in these many steps, else reject.

- If M does not halt on w, it does not halt in $|v|$ steps. So $\mathscr{L}(N) = \Sigma^*$ is infinite.
- M halts on w after s steps. Let $n = |v|$.
  - If $n \geqslant s$, the simulation of M on w halts within n steps, so N rejects v.
  - If $n < s$, the simulation of M on w does not halt in n steps, so N accepts v.

So $\mathscr{L}(N) = \{v \in \Sigma^* \mid |v| < s\}$ which is finite (although dependent on M and w).

Tutorial Exercises

1. Prove that the following languages are not recursive.
   (a) $\{M\,\#\,w \mid M$ writes the blank symbol at some point of time on input $w\}$.
   (b) $\{M\,\#\,w\,\#\,\$ \mid M$ writes the symbol $\$ \in \Gamma$ at some point of time on input $w\}$.

2. (a) Prove that the language $\{M \mid M$ halts on exactly 2021 inputs$\}$ is not r.e.
   (b) Prove that the language $\{M \mid M$ halts on at least 2021 inputs$\}$ is r.e. but not recursive.

3. Let $nsteps(M,w)$ denote the number of steps of M on w. If M loops on w, take $nsteps(M,w) = \infty$. If N also loops on v, take $nsteps(M,w) = nsteps(N,v)$.
   Recursive / r.e. but not recursive / non-r.e.? Prove.
   (a) $\{M\,\#\,N \mid nsteps(M,\varepsilon) < nsteps(N,\varepsilon)\}$.
   (b) $\{M\,\#\,N \mid nsteps(M,\varepsilon) \leqslant nsteps(N,\varepsilon)\}$.
   (c) $\{M\,\#\,N \mid nsteps(M,w) < nsteps(N,v)$ for some $w,v\}$.
   (d) $\{M\,\#\,N \mid nsteps(M,w) < nsteps(N,v)$ for all $w,v\}$.

Tutorial Exercises

4. Prove that the following languages are not recursive.

(a) $\{M \# N \mid \mathscr{L}(M) = \mathscr{L}(N)\}$.

(b) $\{M \# N \mid \mathscr{L}(M) \subseteq \mathscr{L}(N)\}$.

(c) $\{M \# N \mid \mathscr{L}(M) \cap \mathscr{L}(N) = \emptyset\}$.

(d) $\{M \# N \mid \mathscr{L}(M) \cap \mathscr{L}(N) \text{ is finite}\}$.

(e) $\{M \# N \mid \mathscr{L}(M) \cap \mathscr{L}(N) \text{ is regular}\}$.

(f) $\{M \# N \mid \mathscr{L}(M) \cap \mathscr{L}(N) \text{ is context-free}\}$.

(g) $\{M \# N \mid \mathscr{L}(M) \cap \mathscr{L}(N) \text{ is recursive}\}$.

(h) $\{M \# N \# P \mid \mathscr{L}(M) \cap \mathscr{L}(N) = \mathscr{L}(P)\}$.

5. Prove that neither the language $\mathrm{REG} = \{M \mid \mathscr{L}(M) \text{ is regular}\}$ nor its complement is r.e.

6. R.E. or not? Prove.

(a) $\{M \mid M \text{ accepts at most 2021 inputs}\}$.

(b) $\{M \mid M \text{ accepts at least 2021 inputs}\}$.

(c) $\{M \mid M \text{ accepts all strings of length } \leqslant 2021\}$.

(d) $\{M \mid M \text{ does not accept some string of length } \leqslant 2021\}$.