CS423 Finite Automata & Theory of Computation

TTh 12:30 - 13:50 in Smal Physics Lab 111 (section 1)

TTh 9:30 - 10:50 in Blow 331 (section 2)

Prof. Weizhen Mao, wxmaox@wm.edu, wm@cs.wm.edu

General Information

- Office Hours: TTh 11:00 - 12:00 in 114 McGl and W 2:30 - 3:00 on zoom or by email
- Grader: TBD for section 1 (office hour TBD on BB)
- Grader: TBD for section 2 (office hour TBD on BB)
- Textbook: Intro to the theory of computation (any edition), Michael Sipser. An e-book in PDF maybe available online.
- Prerequisites/background: Linear algebra, Data structures and algorithms, and Discrete math

Computability Theory: An introduction

- A study of capability and limitation of computers, or equivalently, what they can do and what they cannot.
- Given a problem, can it be solved at all?
- The set of all problems can be divided into two subsets. One subset contains all solvable problems, such as sorting, finding shortest path in a graph. The other subset contains those unsolvable problems.
- Computability Theory is to study techniques to prove if a given problem is solvable or unsolvable.

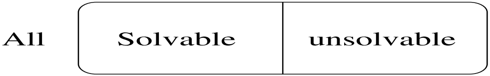


Figure 1: Solvable vs. unsolvable

8.1 Unsolvable problems

- A problem is said to be unsolvable/undecidable if it cannot be solved/decided by any algorithm.
- Most interesting problems are optimization problems (OPT)
- Decision problems (DEC) ask a yes-no question.
- Example: The Traveling Salesman Problem (TSPOPT)
  Visit every city and go back home.
  Input: An edge-weighted graph $G = (V, E, w)$
  Output: A tour (simple cycle of all vertices) with min total weight
- Corresponding decision problem (TSPDEC)
  Input: $G = (V, E, w)$ and $B \geq 0$
  Question: Is there a tour in G with total weight $\leq$ B?

- Meta Claim: DEC is no harder than its corresponding OPT
- So, to study hardness of an OPT, we focus on its DEC.
- Any DEC is actually a language since the yes-no question in DEC can be interpreted as asking membership of a string in a language.
- Example: Prime (DEC)
  Input: An integer $x \geq 2$
  Question: Is x a prime? (This is a yes/no question.)
- $L_{prime} = \{< x > | x \text{ is prime}\}$ (This is a language)
  L_{prime} is actually the language of all prime numbers encoded in binary representation.

- Encoding anything to a binary string:
  - Integer x to binary string $< x >$
  - Graph G to $< G >$
  - Matrix M to $< M >$
  - List L to $< L >$
- Revisit TSPDEC and its corresponding language :
  - Input (or Instance): G and $B \geq 0$
    Question: Does G contain a tour with the total weight $\leq B$?
  - $L_{TSPDEC} = \{< G, B > | \text{There is a tour with total weight } \leq \text{ B}\}$
    Is string $< G, B >$ a member of language L_{TSPDEC}?

- The number of languages over a non-unary alphabet is uncountably infinite. So is the number of DECs (or decision problems).
- However, the number of programs that a computer can use to solve problems is countably infinite. Therefore, there are more problems than there are programs. Thus, there must be some unsolvable problems.

- An unsolvable (or undecidable) problem :
  The famous Halting Problem (by Turing):
  - Input: Any Turing Machine M and any string s
  - Question: Does M halt on s?
- The modern version:
  - Input: Any program P and any input I
  - Output: “Yes” if P terminates on I and “No” otherwise.
    Or Question: Does P terminate on I?

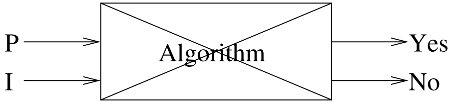


Figure 2: Does P terminate/halt on I?

8.2 Turing machine (*Sipser 3.1, pp. 165-175*)

- A Turing machine includes a control unit, a read-write head, and a one-way infinite tape.

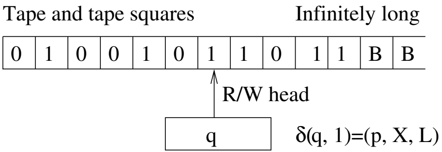


Figure 3: Picture of a Turing Machine

- How to describe a snapshot of a TM without drawing a picture?
  Use a configuration: $010010q11011$

- TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where
  - Q: The finite set of states for the control unit.
  - Σ: An alphabet of input symbols, not containing the "blank symbol", B.
  - Γ: The complete set of tape symbols. $\Sigma \cup \{B\} \subset \Gamma$.
  - δ: The transition function from $Q \times \Gamma$ to $Q \times \Gamma \times D$, where $D = \{L, R\}$.
  - For example, $\delta(q, 0) = (p, X, L)$ and $\delta(p, Y) = (q, B, R)$.
  - q_0: The start state.
  - q_{accept}: The accept state.
  - q_{reject}: The reject state.

- Configuration: Use a string to describe the look of a TM at a certain time, instead of drawing a picture of the TM. For example, string $X_1 \cdots X_{i-1} q X_i \cdots X_n$ gives a description (snapshot) of the TM at a time, when the current state is q, the tape content is $X_1 \cdots X_n$, and the head is scanning (pointing to) X_i. Such a string is called the configuration of the TM at a certain time.

- How a TM changes its configurations:
  - If $\delta(q, X_i) = (p, Y, L)$, then $X_1 \cdots X_{i-1} q X_i \cdots X_n \vdash X_1 \cdots X_{i-2} p X_{i-1} Y X_{i+1} \cdots X_n$.
  - If $\delta(q, X_i) = (p, Y, R)$, then $X_1 \cdots X_{i-1} q X_i \cdots X_n \vdash X_1 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$.

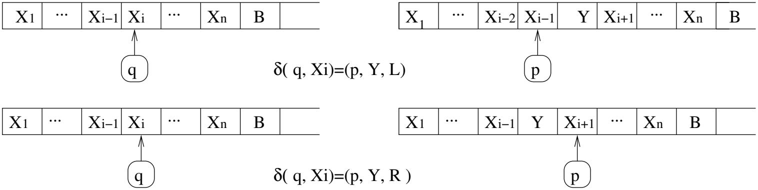


Figure 4: Transitions applied on configurations

- Three important configurations:
  (1) Starting configuration $q_0 w$,
  (2) accepting configuration $uq_{accept}v$,
  (3) rejecting configuration $uq_{reject}v$,
  where (2) and (3) are called the halting configurations.
- Language of a Turing machine M (or language recognized/accepted by M) is
  $L(M) = \{w \in \Sigma^* | q_0 w \overset{*}{\vdash} \alpha q_{accept} \beta \text{ for any } \alpha, \beta \in \Gamma^*\}$.
- Note: To produce $\vdash$, type "backslash vdash" in the math mode.
- For any given input, a TM has three possible outcomes: accept, reject, and loop. Accept and reject mean that the TM halts on the given input, but loop means that the TM does not halt on the input.

- **TRL:** A language A is Turing-recognizable if there is a TM M such that $A = L(M)$. In other words,
  - $\forall w \in A$, M accepts w by entering q_{accept}.
  - $\forall w \notin A$, M does not accept (i.e., it may reject or loop).

- **TDL:** A language A is Turing-decidable if there is a TM M such that $A = L(M)$ and M halts on all inputs. In other words,
  - $\forall w \in A$, M accepts w.
  - $\forall w \notin A$, M rejects w.

  Such TMs are a good model for algorithms.

Figure 5: TRLs vs. TDLs

How to design a TM that recognizes/accepts a language?

Example 1: Give a implementation-level description of a TM M that **accepts** $\{0^n1^n|n \geq 0\}$, i.e., $L(M) = \{0^n1^n|n \geq 0\}$

Idea: $w = 000111 \Rightarrow X00Y11 \Rightarrow XX0YY1 \Rightarrow XXXYYY$

$M =$"On input string $w = 0^n1^n$

1. If $w = \varepsilon$, **accept**
2. Mark the first 0 with X, move right to mark the first 1 with Y
3. Move left to find the leftmost 0. If no 0, **accept**, else go to stage 2"

Example 1: (more) Define a TM that **accepts** $\{0^n1^n \mid n \geq 0\}$

δ	0	1	X	Y	B
q_0	$(q_1, X, R)^1$	-	-	$(q_3, Y, R)^8$	$(q_a, B, R)^0$
q_1	$(q_1, 0, R)^2$	$(q_2, Y, L)^4$	-	$(q_1, Y, R)^3$	-
q_2	$(q_2, 0, L)^6$	-	$(q_0, X, R)^7$	$(q_2, Y, L)^5$	-
q_3	-	-	-	$(q_3, Y, R)^9$	$(q_a, B, R)^{10}$
q_a	-	-	-	-	-

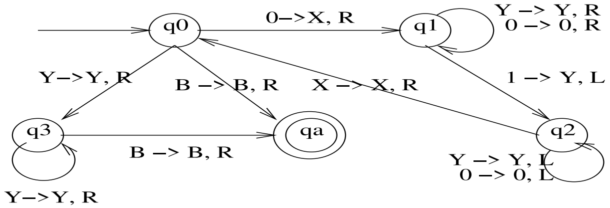


Figure 6: Transition diagram for TM

Example 2: Give an implementation-level description of a TM M that **decides** $\{0^n 1^n | n \geq 0\}$.

$M =$ "On any input string $w \in \{0, 1\}^*$

1. If $w \neq 0^* 1^*$, **reject**
2. Sweep left to right. If no 0 and 1 are found, **accept.** If only 0 is found or 1 is found, but not both, **reject.** If both 0 and 1 are found, go to stage 3
3. Mark the leftmost 0 with X. Move head to right to find and mark the first 1 with Y
4. Move head to left end, and then go to stage 2"

Example 3.7 (Sipser p. 171): Give a TM M that **decides** $A = \{0^{2^n} | n \geq 0\} = \{0, 00, 0000, 00000000, \cdots\}$.

Consider the following strings to figure out an algorithm (TM):
(1) odd length, e.g., $w_1 = 00000 \Rightarrow 0X0X0$;
(2) even length, e.g.,
$w_2 = 00000000 \Rightarrow 0X0X0X0X \Rightarrow 0XXX0XXX \Rightarrow 0XXXXXXX$;
$w_3 = 000000 \Rightarrow 0X0X0X$

TM $M =$ ”On input string $w \in \{0\}^*$:

1. Sweep left to right, crossing off every other 0
2. If in stage 1 the tape contained a single 0, **accept** (e.g., w_2)
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, **reject** (e.g., w_1 and w_3)
4. Move head to the left end of the tape
5. Go to stage 1”

8.4 Variations of TMs (*Sipser 3.2 (pp. 148-159)*)

- TM with multi-tapes (and multi-heads) ($\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$).
- TM with multi-strings (and multi-heads).
- TM with multi-heads.
- TM with multi-tracks.
- TM with two-way infinite tape.
- TM with multi-dimensional tape.
- Nondeterministic TM's ($\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times D}$).
  Consider a move in NTM, $\delta(q_3, X) = \{(q_5, Y, R), (q_3, X, L)\}$.
  How does the NTM know which step it should take? One way to look at this is: the NTM is the ”luckiest possible guesser” and it always picks a transition that eventually leads to an accepting state if there is such a transition.

Theorem: The equivalent **computing power** of the above TM's:

For any language L, if $L = L(M_1)$ for some TM M_1 with multi-tapes, multi-strings, multi-heads, multi-tracks, two-way infinite tape, multi-dimensional tape, or nondeterminism, then $L = L(M_2)$ for some basic TM M_2.

Theorem: The equivalent **computing speed** of the above TM's except for nondeterministic TM's:

For any language L, if $L = L(M_1)$ for some TM M_1 with multi-tapes, multi-strings, multi-heads, multi-tracks, two-way infinite tape, or multi-dimensional tape in a polynomial number of steps, then $L = L(M_2)$ for some basic TM M_2 in a polynomial number of steps (with a higher degree).

Or in other words, all reasonable models of computation can simulate each other with only a polynomial loss of efficiency.

Note: The speed-up of a nondeterministic TM vs. a basic TM is exponential.

The Church-Turing Thesis:

Any reasonable attempt to model mathematically algorithms and their time performance is bound to end up with a model of computation and associated time cost that is equivalent to Turing machines within a polynomial. (The power of TM.)

Nondeterministic TMs

- $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times D}$.
- Consider a move in NTM, $\delta(q_3, X) = \{(q_5, Y, R), (q_3, X, L)\}$. How does the NTM know which step it should take?
- One way to look at this is that the NTM is the ”luckiest possible guesser” and it always picks a transition that eventually leads to an accepting state if there is such a transition.
- The other way is to imagine that the NTM branches into many copies, each of which follows one of the possible transition.
- DTM (path) versus NTM (tree): See the wiki page for ”Nondeterministic Turing Machine”.

Theorem: A TDL is also a TRL, but not vice versa.

Theorem: About A and $\overline{A}$:

1. If A is Turing-decidable, so is $\overline{A}$.
2. If A and $\overline{A}$ are both Turing-recognizable, then A is Turing-decidable. (See Theorem 4.22, p.210)
3. For any A and $\overline{A}$, we have one of the following possibilities:
   (1) Both are Turing-decidable;
   (2) Neither is Turing-recognizable;
   (3) One is Turing-recognizable but not decidable, the other is not Turing-recognizable.

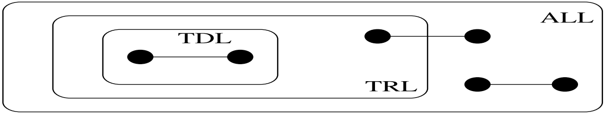


Figure 7: A language and its complement

Some closure properties:

TRLs and TDLs are both closed under

- Union
- Intersection
- Concatenation
- Star

In addition, TDLs are closed under complement, and TRLs are closed under homomorphism.

Examples to prove closure properties:

Example 1: If L_1 and L_2 are TD, so is $L_1 \cup L_2$.

Pf: Let TM M_1 and TM M_2 decide L_1 and L_2, respectively. Then we have the following TM M to decide $L_1 \cup L_2$.

TM $M =$ "On input w:

1. Run M_1 on w
2. If M_1 accepts, **accept**
3. else run M_2 on w
4. If M_2 accepts, **accept**
5. else **reject**"

Example 2: If L_1 and L_2 are TR, so is $L_1 \cup L_2$.

Pf: Let TM M_1 and TM M_2 recognize L_1 and L_2, respectively. Then we have the following TM M to recognize $L_1 \cup L_2$.

TM $M =$ "On input $w \in L_1 \cup L_2$

1. Run M_1 and M_2 alternately on w, one step at a time
2. If either accepts, **accept**"

Example 3: If L is TD, so is L^*.

Pf: Let TM M decide L. Then we have the following TM M^* to decide L^*.
Note: $w \in L^*$ if $w = w_1 w_2 \cdots w_k$ for some $k \in [1, |w|]$, where $w_i \in L$ for $i = 1, \cdots, k$.

TM $M^* =$ "On input w

1. If $w = \varepsilon$, **accept**
2. $\forall k = 1, 2, \cdots, |w|$
3. $\forall$ partitions of w into k substrings, i.e., $w_1, w_2, \cdots, w_k$
4. Run M on $w_1, w_2, \cdots, w_k$
5. If M accepts w_i, $\forall i = 1, \cdots k$, **accept**
6. **reject**"

Example 4: If L is TR, so is L^*
Pf: Let TM M recognize L. Then we have the following NTM N to recognize L^*

NTM $N =$ "On input $w \in L^*$

1. Nondeterministically generate (guess) a partition of w into $w_1, w_2, \cdots, w_k$
2. Run M on $w_1, w_2, \cdots, w_k$
3. If M accepts w_i, $\forall i = 1, 2, \cdots, k$, **accept**"

9.1 A binary encoding scheme for TMs

- TM $\Leftrightarrow$ binary number.
  $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$ with q_1 to be the start state, q_2 to be the accept state, and q_3 to be the reject state.
  $\Gamma = \{X_1, X_2, \ldots, X_{|\Gamma|}\}$.
  $D = \{D_1, D_2\}$ with D_1 to be L and D_2 to be R.
  A transition $\delta(q_i, X_j) = (q_k, X_l, D_m)$ is coded as $0^i 1 0^j 1 0^k 1 0^l 1 0^m$.
  A TM is coded as $C_1 11 C_2 11 \cdots 11 C_n$, where each C is the code for a transition.
- An example: $\delta(q_2, X_3) = (q_1, X_4, D_1)$ can be coded as 001000101000010
- An example: 000010010100100 is the encoding of $\delta(q_4, X_2) = (q_1, X_2, D_2)$

- TM M with input w is represented by $< M, w >$ and encoded as $< M > 111w$.
- Using similar schemes, we can encode DFA, NFA, PDA, RE, and CFG into binary strings.

9.2 Decidable languages (Sipser 4.1, pp. 194-201)

- $A_{DFA} = \{< B, w > | B \text{ is a DFA that accepts string } w\}$.
- $A_{NFA} = \{< B, w > | B \text{ is an NFA that accepts string } w\}$.
- $A_{REX} = \{< R, w > | R \text{ is a RE that generates string } w\}$.
- $E_{DFA} = \{< B > | B \text{ is a DFA and } L(B) = \emptyset\}$.
- $EQ_{DFA} = \{< B_1, B_2 > | B_1 \text{ and } B_2 \text{ are DFAs and } L(B_1) = L(B_2)\}$.
- $A_{CFG} = \{< G, w > | G \text{ is a CFG that generates string } w\}$.
- $E_{CFG} = \{< G > | G \text{ is a CFG and } L(G) = \emptyset\}$.
- Every CFL is decidable.

Note: The proofs of these TDLs can be found in Sipser's book.

Example 1. Prove that $A_{DFA} = \{< B, w > | w \in L(B)\}$ is TD.

TM M = On input $< B, w >$
Simulate B on input w
If B ends in $q \in F$, **accept**
else **reject**

Example 2. Prove that $E_{DFA} = \{< B > | L(B) = \emptyset\}$ is TD.

TM M = On input $< B >$
Create the state diagram G for B
Use DFS to generate all simple paths from q_0 to any $q \in F$
If no path is found, **accept**
else **reject**

Countable and uncountable sets

- The size of an infinite set: Countably infinite (or countable) and uncountably infinite (or uncountable).
- A set A is countable if there is a 1-1 correspondence with $N = \{1, 2, 3, \ldots\}$ (the set of natural numbers).
- The following sets are countable.
  1. The set of even (or odd) numbers
  2. The set of rationale numbers
  3. The set of binary strings
  4. The set of TMs
- But, the set of languages is uncountable.
- There are more languages than there are TMs. So there must be languages that are non-TRL.

9.4 A non-TRL

- Consider the binary alphabet.
- Order and label strings: $\varepsilon, 0, 1, 00, 01, 10, 11, \cdots$.
  Let w_i be ith string in the above lexicographic ordering.
- Order and label TMs: $M_1, M_2, M_3, \cdots$.
  Let M_i be the TM whose code is w_i, i.e. $< M_i >= w_i$.
  In case w_i is not a valid TM code, let M_i be the TM that immediately rejects any input, i.e., $L(M_i) = \emptyset$.

ε	0	1	00	01	10	11	000	$\cdots$
w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	$\cdots$
M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	$\cdots$

- For any string w_i, there is a TM M_i
- For any TM M_i, there is a string w_i, where $< M_i >= w_i$.

- Define diagonalization language $A_D = \{w_i | w_i \notin L(M_i)\}$.
- The corresponding decision problem:
  Input: Any binary string w_i
  Question: Is w_i not accepted by M_i?
- Prove that A_D is non-TR (not a TRL).
  Proof:
  1. Suppose, by contradiction, A_D is TR, i.e., there is a TM M such that $A_D = L(M)$.
  2. Then $M = M_i$ with code w_i for some i.
  3. $w_i \in A_D$ iff $w_i \notin L(M_i)$ by definition of A_D.
  4. $w_i \in A_D$ iff $w_i \in L(M_i)$ by $A_D = L(M_i)$.
  5. A contradiction within the two iff statements

  .

A summary of some new concepts learned recently

- Encoding of TM, DFA, NFA, PDA, RE, Graph, Matrix, list, etc. to binary strings: e.g., $< M >$, $< M, w >$, $< M_1, M_2 >$
- Infinite sets: Countable vs. uncountable. Compare the set of TMs (countable) vs. the set of languages (uncountable). There are languages without a TM to accept/recognize.
- Correspondence between binary strings and Turing machines, i.e., for any w_i, there is a M_i and for any M, there is i s.t. $< M >= w_i$. Thus M can be renamed as M_i
- The diagonalization language $A_D = \{w_i \mid w_i \notin L(M_i)\}$ non-TR.

9.5 A TRL but non-TDL (*Sipser 4.2 (pp. 173-174 and 179-182)*)

- A universal TM:
  - Each TM (among those discussed) can only solve a single problem, however, a computer can run arbitrary algorithms. Can we design a general-purposed TM that can solve a wide variety of problems just as a computer?
  - Theorem: There is a universal TM U which simulates an arbitrary TM M with input w and produces the same output.
    TM $U =$ "On input $< M, w >$
    Run M on w"
  - TM U is an abstract model for computers just as TM M is a formal notion for algorithms.

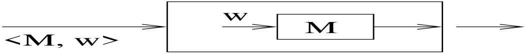


Figure 8: The universal Turing machine

- Let $A_{TM} = \{< M, w > | M \text{ accepts string } w\}$
  Or equivalently $A_{TM} = \{< M, w > | w \in L(M)\}$
  Or equivalently as a decision problem
  Input: A TM M and a string w
  Question: Is w accepted by M?

  A_{TM} is called the universal language.

  A_{TM} is TR since it can be recognized by TM U.

  TM $U =$ "On input $< M, w > \in A_{TM}$
  Run M on w
  If M accepts w, **accept**"

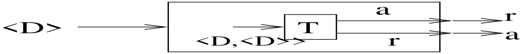
- $A_{TM} = \{< M, w > | w \in L(M)\}$ is non-TD. (By C&C)
  1. Assume that A_{TM} is decided by TM T.

     Input to T is $< M, w >$
     Output from T is **accept** if $w \in L(M)$ and **reject** if $w \notin L(M)$
  2. On input $< M, w >$, T accepts $< M, w >$ iff M accepts w. (We can also say, T rejects $< M, w >$ iff M rejects w.)
  3. Define TM D as follows:

  4. Observe that D accepts $< M >$ iff T rejects $< M, < M >>$.
  5. Feed $< D >$ to D.

  6. From steps 4 and 2, D accepts $< D >$ iff T rejects $< D, < D >>$ iff D rejects $< D >$.
  7. A contradiction in step 6.

- Another proof that $A_{TM} = \{< M, w > | w \in L(M)\}$ is non-TD.
- Proof. Assume that A_{TM} is TD by TM T, i.e.,
- Let TM T decide A_{TM}, i.e.,

$$T(< M, w >) = \begin{cases} \textbf{accept} & w \in L(M) \\ \textbf{reject} & w \notin L(M) \end{cases}$$

- Define TM $D =$"On input $< M >$
  Run T on $< M, < M >>$
  If T accepts, **reject**
  If T rejects, **accept**"
- D accepts $< M >$ iff T rejects $< M, < M >>$
- Feed $< D >$ to D. Then, D accepts $< D >$ iff T rejects $< D, < D >>$ iff $< D > \notin L(D)$ iff D rejects $< D >$.
- A contradiction! So A_{TM} is non-TD.

10.1 A summary of terminology in Computability Theory

- Language, Decision Problem, Problem
- TM, Algorithm, Solution
- Decide, Solve, (Decidable, Solvable)
- Undecidable, Unsolvable
- Accept, Recognize, (Acceptable, Recognizable)

10.2 A review of some languages and corresponding decision problems

- $A_D = \{w_i | w_i \not\in L(M_i)\}$ (non-TR)
  Input: Any binary string w_i
  Question: Is w_i not accepted by M_i?
- $A_{TM} = \{< M, w > | w \in L(M)\}$ (TR but non-TD)
  Input: TM M and string w
  Question: Does M accept w?
  It is undecidable whether TM M accepts string w for any given M and w.
- $HALT_{TM} = \{< M, w > | M \text{ halts on } w\}$ (TR but non-TD)
  Input: TM M and string w
  Question: Does M halt on w?
  It is undecidable whether TM M halts string w for any given M and w.

10.3 Reducibility or Reduction(*Sipser 5 (pp. 216-220)*)

We say that problem A reduces (or is reducible) to problem B, written as $A \leq B$, if we can use a solution (TM) to B to solve A (i.e., if B is decidable/solvable, so is A.).
We may use reducibility to prove undecidability as follows:

1. Let A be non-TD, such as A_D or A_{TM}. Wish to prove B is non-TD.
2. Assume B is TD. Then there exists a TM M_B to decide B.
3. If we can use M_B as a sub-routine to construct a TM M_A that decides A, then A is TD. We have a contradiction.
4. The construction of TM M_A using TM M_B establishes that A reduces to B, i.e., $A \leq B$. (A is no harder than B)
5. Corollary 5.23 (Sipser p. 236):
   If $A \leq B$ and A is non-TD, then B is non-TD.

10.4 A proof of the non-TD A_{TM} by reduction

- Proof sketch:
  1. Assume A_{TM} is TD, by contradiction.
  2. Let TM S decide A_{TM}, by the definition of a TDL.
  3. Try to construct a TM D that decides A_D. The construction will include TM S. This shows A_D is TD.
  4. A contradiction since we know A_D is non-TR.

  Note: This proof uses the TM S for A_{TM} to build a TM D for A_D, i.e., $A_D \leq A_{TM}$.
- Recall two languages:
  1. $A_{TM} = \{< M, w > | w \in L(M)\}$.
  2. $A_D = \{w_i | w_i \notin L(M_i)\}$. ($A_D$ is non-TR)

Prove that A_{TM} is non-TD by reduction

1. Assume A_{TM} is TD, by contradiction.
2. Let TM S decide A_{TM}, i.e.,

$$S(< M, w >) = \begin{cases} \textbf{accept} & w \in L(M) \\ \textbf{reject} & w \notin L(M) \end{cases}$$

3. Construct a TM D that decides A_D, a non-TRL.
   TM $D =$"On input w_i
   Run S on $< M_i, w_i >$
   If S accepts, **reject** else **accept**"
4. Why does D decide A_D?
   S accepts $< M_i, w_i >$ *iff* $w_i \in L(M_i)$ iff $w_i \notin A_D$ iff D rejects w_i. So S accepts iff D rejects.
5. So A_D is TD. A contradiction.

10.5 The halting problem (*Theorem 5.1 (pp. 216-217)*)

- $HALT_{TM} = \{< M, w > | M \text{ halts on string } w\}$.
- $A_{TM} = \{< M, w > | M \text{ accepts } w\}$
- $A_{TM} \subseteq HALT_{TM}$
- $HALT_{TM}$ is TR since it can be recognized by TM U.
- **Theorem 5.1** $HALT_{TM}$ is non-TD.
  (Will show $A_{TM} \leq HALT_{TM}$)

Theorem $HALT_{TM}$ is non-TD. Prove by reduction from A_{TM}, i.e., $A_{TM} \leq HALT_{TM}$

1. Assume TM R decides $HALT_{TM}$. Then R accepts $< M, w >$ iff M halts on w. Construct TM S to decide A_{TM}.

   TM $S =$ "On input $< M, w >$
   Run R on $< M, w >$
   if R rejects, **reject**
   if R accepts, run M on w until it halts
           if M accepts, **accept**; else **reject**"

2. Why does S accept A_{TM}?
   R rejects $< M, w > \Rightarrow M$ doesn't halt on $w \Rightarrow M$ doesn't accept $w \Rightarrow < M, w > \notin A_{TM} \Rightarrow S$ rejects
   R accepts $< M, w > \Rightarrow M$ halts on w (accepts or rejects? Need to run M on w to find out)
   M accepts $w \Rightarrow < M, w > \in A_{TM}$

3. Since we constructed a TM S that decides A_{TM} using TM R, so A_{TM} is TD. A contradiction to that A_{TM} is proved to be non-TD.

10.6 Other non-TD problems (*Sipser 5.1 (pp. 216-220)*)

The following problems about Turing machines are non-TD:

- Whether $L(M) = \emptyset$ for any TM M.
  $E_{TM} = \{< M > | L(M) = \emptyset\}$
  $NE_{TM} = \{< M > | L(M) \neq \emptyset\}$ (complement of E_{TM})
- Whether $L(M_1) = L(M_2)$ for any two TMs M_1 and M_2.
  $EQ_{TM} = \{< M_1, M_2 > | L(M_1) = L(M_2)\}$
- Whether $L(M)$ is finite for any TM M
  $FINITE_{TM} = \{< M > | \text{L(M) is finite}\}$
- Whether $\varepsilon \in L(M)$ for any TM M.
  $ESTRING_{TM} = \{< M > | \varepsilon \in L(M)\}$
- Whether $L(M) = \Sigma^*$ for any TM M.
  $ALL_{TM} = \{< M > | L(M) = \Sigma^*\}$

Rice's Theorem: Every nontrivial property of the TRLs (or TMs) is undecidable.

Pf: $E_{TM} = \{< M > | L(M) = \emptyset\}$ is non-TD. Let R decides E_{TM}.

$$R(< M >) = \begin{cases} \textbf{accept} & L(M) = \emptyset \\ \textbf{reject} & L(M) \neq \emptyset \end{cases}$$

(2) Use R to construct TM S that decides A_{TM}, i.e., $A_{TM} \leq E_{TM}$.
TM $S =$ "On input $< M, w >$,

- Construct TM $M_1 =$ "On input x
  If $x \neq w$ reject else run M on w"
  Note: $L(M_1) = \{w\}$ if $w \in L(M)$; $L(M_1) = \phi$ if $w \notin L(M)$
- Run R on $< M_1 >$
- If R accepts, **reject**; and if R rejects, **accept**"

(3) Why does S decide A_{TM}? $L(M_1) = \emptyset$ if M does not accept w; and $L(M_1) = \{w\}$ if M accepts w. I.e., $L(M_1) = \emptyset$ iff $w \notin L(M)$. So R accepts $< M_1 >$ iff $L(M_1) = \emptyset$ iff $w \notin L(M)$ iff S rejects.
(4) TM S decides the non-TD A_{TM}. A contradiction.

A graphical explanation of the undecidability proof of E_{TM}

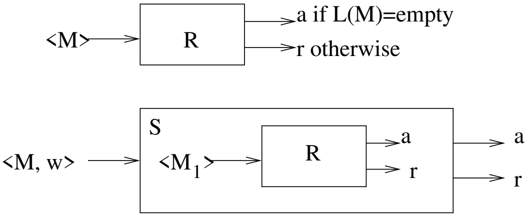


Figure 9: Reduction from A_{TM} to E_{TM}

Important questions to answer:

- Input: how to define M_1 (the input to R) using $< M, w >$ (the input to S)?
- Output: how the output from R implies the output from S?

Goal: Design M_1 such that the output from R defines that of S.

Prove that $NE_{TM} = \{< M > | L(M) \neq \emptyset\}$ is TR but non-TD.
(1) To prove NE_{TM} is TR, we give a NTM N to recognize NE_{TM}.
NTM $N =$ "On input $< M > \in NE_{TM}$

- Guess a string w
- Run M on w
- If M accepts, **accept**"

We can also use a deterministic TM to recognize NE_{TM}.
TM $D =$ "On input $< M > \in NE_{TM}$
Recall the binary sequence $w_1, w_2, w_3, \ldots$

- Systematically generates strings: ε, 0, 1, 00, 01, ...
- for $i = 1, 2, 3, \ldots$
  Run M on $w_1, \cdots, w_i$, each for i steps
- If in the loop above, M ever accepts some w_j, then **accept"**

An explanation of the TM D that recognizes NE_{TM}:

Assume w_9 is accepted by M in 7 steps.
Assume w_{10} is accepted by M in 12 steps.

$i = 1$: Run M on w_1 for 1 step;
$i = 2$: Run M on w_1, w_2 each for 2 steps;
$i = 3$: Run M on w_1, w_2, w_3 each for 3 steps;
$\cdots\cdots$
$i = 9$: Run M on $w_1, w_2, \cdots, w_9$ for 9 steps; (accepted)
$\cdots\cdots$
$i = 12$: Run M on $w_1, w_2, \cdots, w_{10}, \cdots, w_{12}$ for 12 steps (accepted)

(2) To prove $NE_{TM} = \{< M > | L(M) \neq \emptyset\}$ is non-TD, assume it is decided by TM R. Then R accepts $< M >$ iff $L(M) \neq \emptyset$. Construct a TM S that decides the undecidable A_{TM}. Then a contradiction.

TM $S =$ "On input $< M, w >$

1. Construct TM M_1="On input x
   If $x \neq w$, reject else Run M on w"
   Note: $L(M_1) = \{w\}$ if $w \in L(M)$; $L(M_1) = \phi$ if $w \notin L(M)$
2. Run R on $< M_1 >$
3. If R accepts, **accept**; else **reject**"

Why does S accept A_{TM}?

$L(M_1) = \emptyset$ if $w \notin L(M)$ and $L(M_1) = \{w\}$ if $w \in L(M)$. In other words, $L(M_1) \neq \emptyset$ iff $w \in L(M)$.

R accepts $< M_1 >$ iff $L(M_1) \neq \emptyset$ iff $w \in L(M)$ iff $< M, w > \in A_{TM}$ iff S accepts $< M, w >$. So A_{TM} is TD. A contradiction.

About E_{TM} and its complement NE_{TM}

We proved: E_{TM} is non-TD. NE_{TM} is TR.

Recall the theorem on page 120 . For A and $\overline{A}$,

1. Both are TD; (Both are TR)
2. Neither is TR;
3. One is TR but non-TD, the other is non-TR

We immediately have the following results.

(1) NE_{TM} is non-TD (If NE_{TM} is TD, so is E_{TM})

(2) E_{TM} is non-TR (If E_{TM} is TR, both E_{TM} and NE_{TM} are TD)

Theorem 5.4 $EQ_{TM} = \{< M_1, M_2 > | L(M_1) = L(M_2)\}$ is non-TD.
Reduce from $E_{TM} = \{< M > | L(M) = \emptyset\}$.

1. Assume EQ_{TM} is decided by TM R.

$$R(< M_1, M_2 >) = \begin{cases} \textbf{accept} & L(M_1) = L(M_2) \\ \textbf{reject} & L(M_1) \neq L(M_2) \end{cases}$$

2. Construct TM S that decides the undecidable E_{TM}.
   TM $S =$"On input $< M >$
   Construct TM $M_1 =$"On input x, reject"
   Run R on $< M_1, M >$
   R accepts $< M_1, M >$ iff $\emptyset = L(M)$ iff S accepts $< M >$
   R rejects $< M_1, M >$ iff $\emptyset \neq L(M)$ iff S rejects $< M >$
3. Why does S decides E_{TM}? R accepts $< M_1, M >$ iff $L(M_1) = L(M)$ iff $L(M) = \emptyset$ iff S accepts $< M >$.
4. S decides E_{TM}. So E_{TM} is TD. A contradiction.

10.7 Post's correspondence problem (PCP) (*Sipser 5.2*)

INPUT: $P = \{\frac{t_1}{b_1}, \frac{t_2}{b_2}, \ldots, \frac{t_k}{b_k}\}$, where $t_1, t_2, \ldots, t_k$ and $b_1, b_2, \ldots, b_k$ are strings over alphabet Σ. (P is a collection of dominos, each containing two strings, with one stacked on top of the other.)

QUESTION: Does P contain a match?
Or, is there $i_1, i_2, \ldots, i_l \in \{1, 2, \ldots, k\}$ with $l \geq 1$ such that $t_{i_1} t_{i_2} \cdots t_{i_l} = b_{i_1} b_{i_2} \cdots b_{i_l}$?

Equivalently, defined as a language, we have
$L_{PCP} = \{< P > | P \text{ is an instance of PCP with a match}\}$.

For input $P_1 = \{\frac{b}{ca}, \frac{a}{ab}, \frac{ca}{a}, \frac{abc}{c}\}$, sequence $2, 1, 3, 2, 4$ indicates a match. Since $\frac{a}{ab}\ \frac{b}{ca}\ \frac{ca}{a}\ \frac{a}{ab}\ \frac{abc}{c}$, top=bottom=abcaaabc

For $P_2 = \{\frac{abc}{ab}, \frac{ca}{a}, \frac{acc}{ba}\}$, there is no match since all top strings are longer than bottom strings..

PCP is non-TD for the binary alphabet.

A Summary of Computability Theory

1. Definitions and concepts:
   - Turing machine, how it works, its language, its encoding, Church-Turing Thesis
   - TRL and TDL, properties, how M accepts/decides a language, implementation-level description
   - Reduction, the meaning of $A \leq B$ (A is no harder than B), use reduction to prove undecidability
2. Various proofs:
   - A language is TR/TD (prove by definition)
   - A language is non-TR/non-TD (prove by a combination of contradiction, construction, and reduction)
   - Many examples to learn from