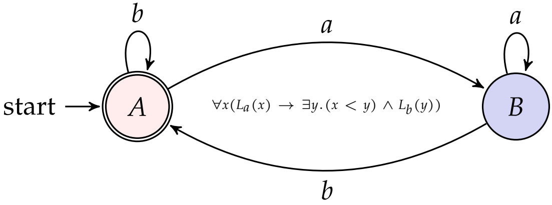
CS 208: Automata Theory and Logic

Lecture 7: Turing Machines

Ashutosh Trivedi



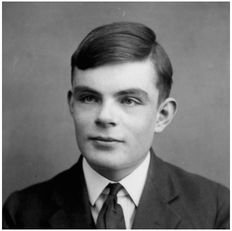
Department of Computer Science and Engineering,
Indian Institute of Technology Bombay.

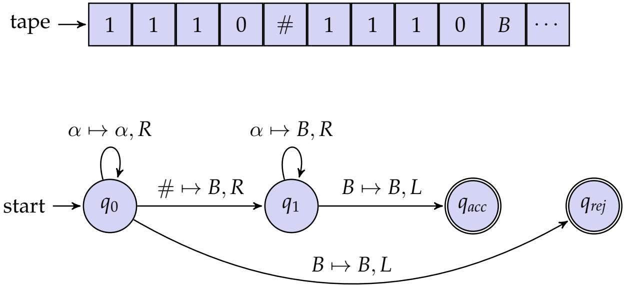
Turing Machines

Undecidability

Reductions

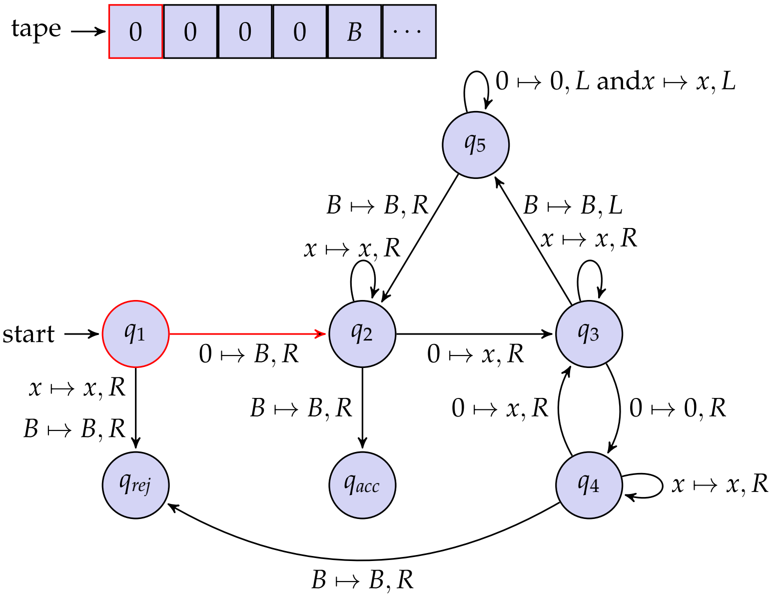
Turing Machine



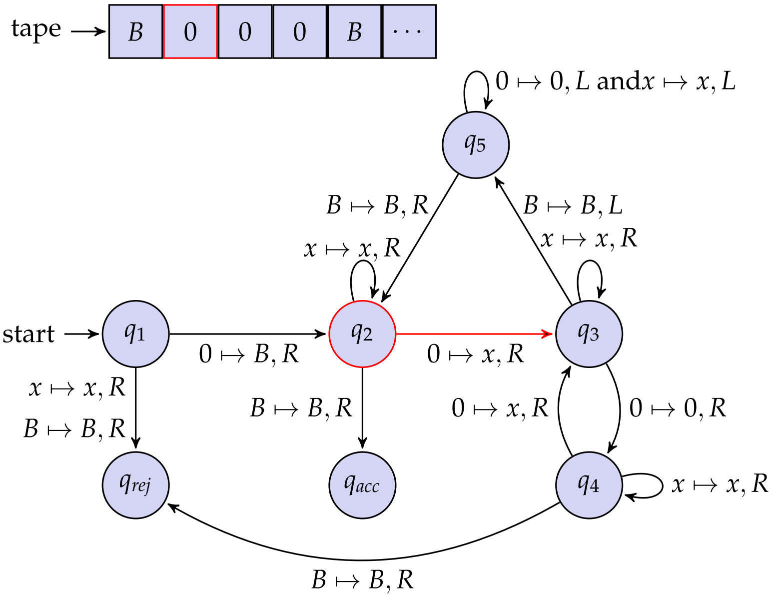


- David Hilbert in 1928 posed the famous Entschiedungsproblem of finding an effective computation (Algorithm) to decide using a finite number of operations whether a given FO-formula is valid.
- Kurt Gödel in 1931, via his famous Incompleteness Theorem abstractly answered this question by proving that there is no "effective computation" to solve all mathematical questions.
- Alan Turing formalized the notion of "effective computation" using Turing machines, formalized the notion of undecidability, and proved the Entschiedungsproblem to be undecidable.

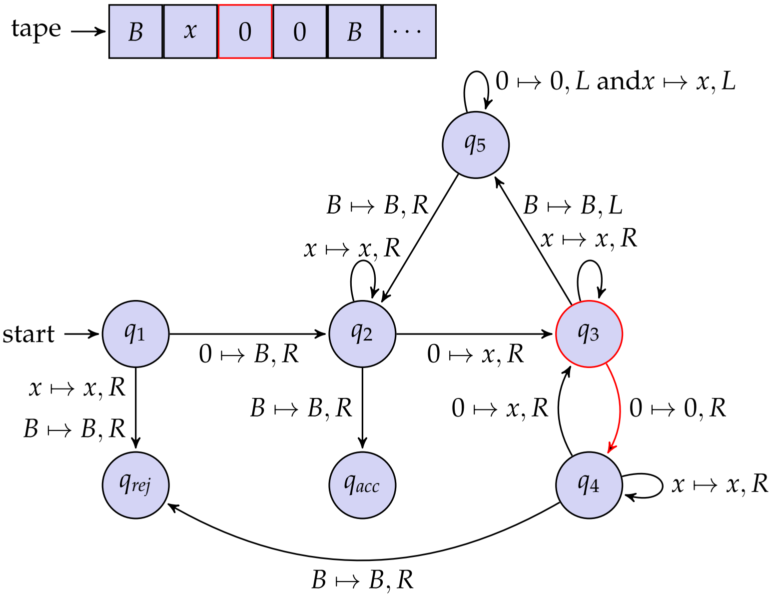
Example 1: $L = \{0^{2^n} : n \geq 0\}$



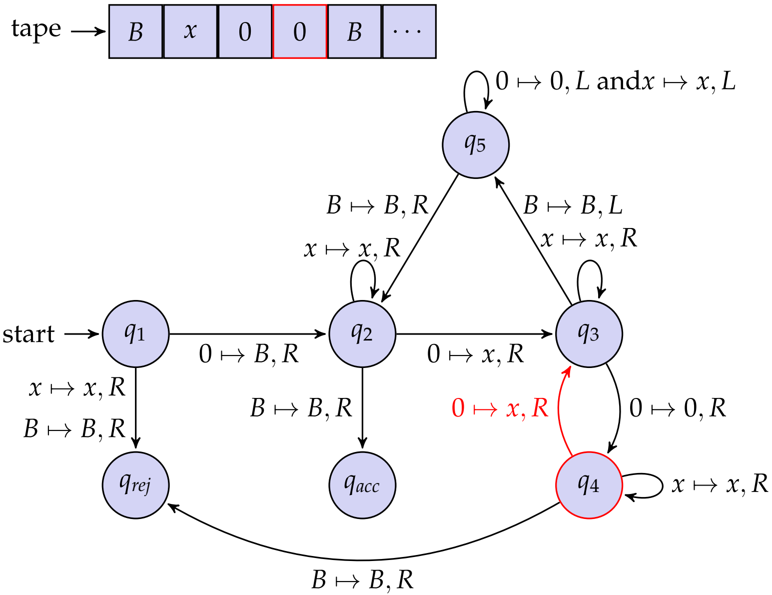
Example 1: $L = \{0^{2^n} : n \geq 0\}$



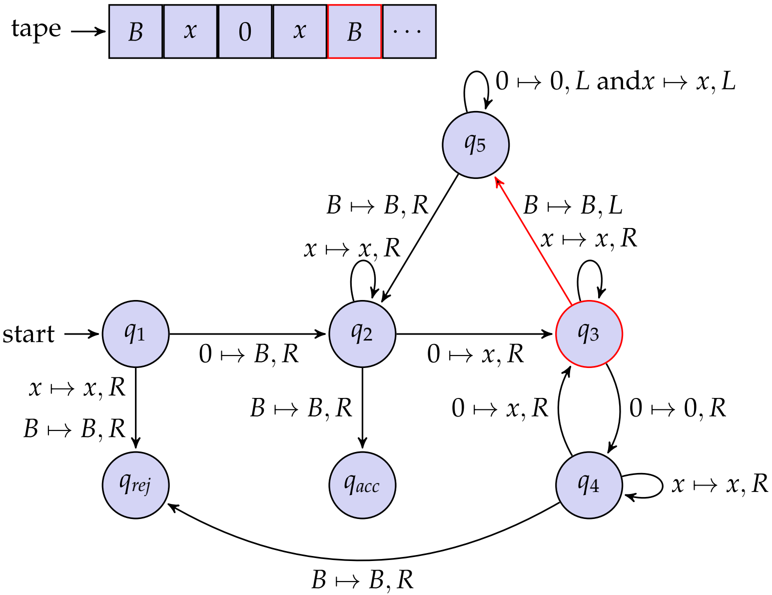
Example 1: $L = \{0^{2^n} : n \geq 0\}$



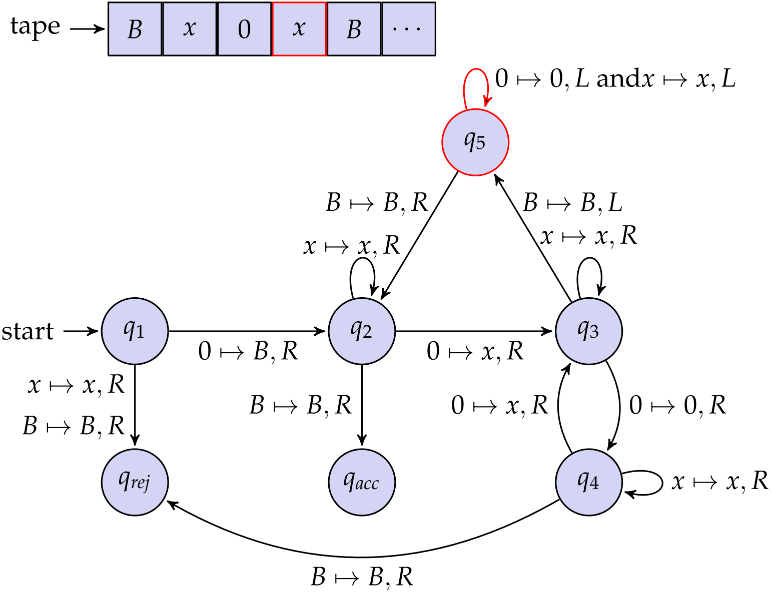
Example 1: $L = \{0^{2^n} : n \geq 0\}$



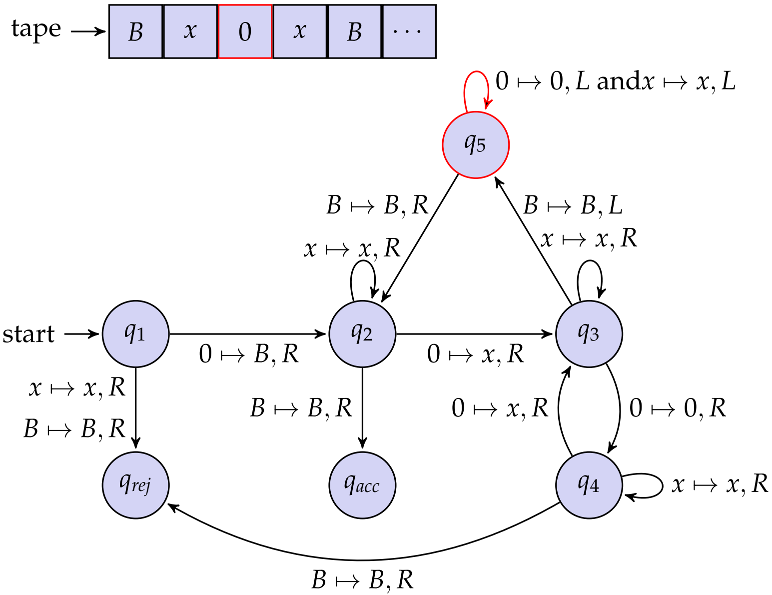
Example 1: $L = \{0^{2^n} : n \geq 0\}$



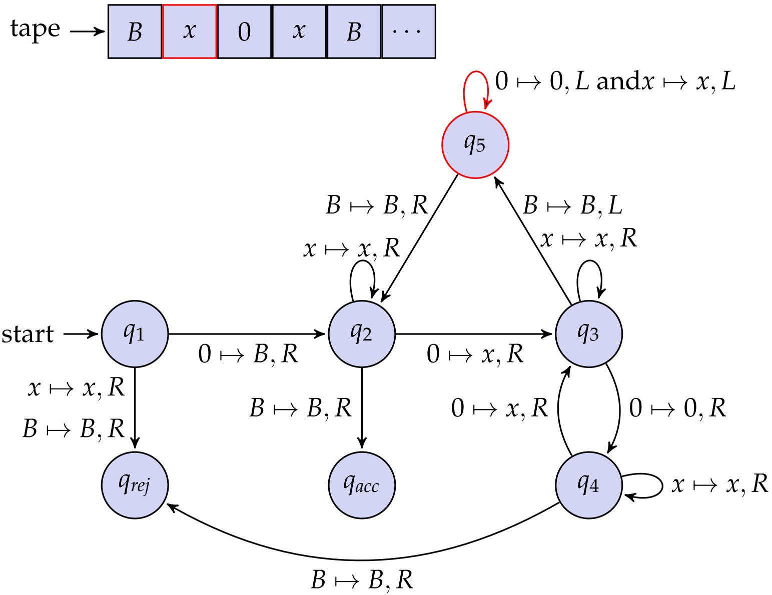
Example 1: $L = \{0^{2^n} : n \geq 0\}$



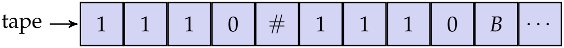
Example 1: $L = \{0^{2^n} : n \geq 0\}$

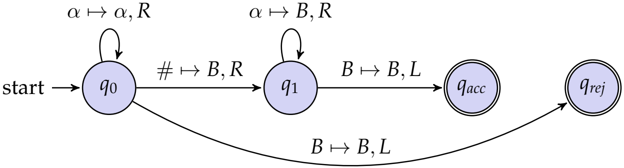


Example 1: $L = \{0^{2^n} : n \geq 0\}$



Turing Machines





A Turing machine is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}$ where:

- Q is a finite set called the states;
- Σ is a finite set called the alphabet not containing blank symbol B;
- Γ is a finite set called the tape alphabet, where $B \in \Gamma$ and $\Sigma \subseteq \Gamma$;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function;
- $q_0 \in Q$ is the start state;
- $q_{acc} \in Q$ is the accept state, and
- $q_{rej} \in Q$ is the reject state, where $q_{acc} \neq q_{rej}$.

Semantics of Turing Machines

- A configuration is a tuple $(q, u, v) \in Q \times \Gamma^* \times \Gamma^*$ where
  1. q is the current state,
  2. u is the string on the tape to the left of the tape head, and
  3. v is the string to the right of the tape head, and tape head is pointing to the first symbol of v.
- We write a configuration as $\langle u, q, v \rangle$.

Semantics of Turing Machines

- A configuration is a tuple $(q, u, v) \in Q \times \Gamma^* \times \Gamma^*$ where
  1. q is the current state,
  2. u is the string on the tape to the left of the tape head, and
  3. v is the string to the right of the tape head, and tape head is pointing to the first symbol of v.
- We write a configuration as $\langle u, q, v \rangle$.
- $\langle \varepsilon, q_0, w \rangle$ is the start configuration
- $\langle u, q_{acc}, w \rangle$ is the accepting configuration, and
- $\langle u, q_{rej}, w \rangle$ is the rejecting configuration.

Semantics of Turing Machines

- A configuration is a tuple $(q, u, v) \in Q \times \Gamma^* \times \Gamma^*$ where
  1. q is the current state,
  2. u is the string on the tape to the left of the tape head, and
  3. v is the string to the right of the tape head, and tape head is pointing to the first symbol of v.
- We write a configuration as $\langle u, q, v \rangle$.
- $\langle \varepsilon, q_0, w \rangle$ is the start configuration
- $\langle u, q_{acc}, w \rangle$ is the accepting configuration, and
- $\langle u, q_{rej}, w \rangle$ is the rejecting configuration.
- If $\delta(q_i, b) = (q_j, c, R)$ then $\langle ua, q_i, bv \rangle$ yields $\langle uac, q_j, v \rangle$, and

Semantics of Turing Machines

- A configuration is a tuple $(q, u, v) \in Q \times \Gamma^* \times \Gamma^*$ where
  1. q is the current state,
  2. u is the string on the tape to the left of the tape head, and
  3. v is the string to the right of the tape head, and tape head is pointing to the first symbol of v.
- We write a configuration as $\langle u, q, v \rangle$.
- $\langle \varepsilon, q_0, w \rangle$ is the start configuration
- $\langle u, q_{acc}, w \rangle$ is the accepting configuration, and
- $\langle u, q_{rej}, w \rangle$ is the rejecting configuration.
- If $\delta(q_i, b) = (q_j, c, R)$ then $\langle ua, q_i, bv \rangle$ yields $\langle uac, q_j, v \rangle$, and
- If $\delta(q_i, b) = (q_j, c, L)$ then $\langle ua, q_i, bv \rangle$ yields $\langle u, q_j, acv \rangle$.

Semantics of Turing Machines

– A configuration is a tuple $(q, u, v) \in Q \times \Gamma^* \times \Gamma^*$ where
  1. q is the current state,
  2. u is the string on the tape to the left of the tape head, and
  3. v is the string to the right of the tape head, and tape head is pointing to the first symbol of v.
– We write a configuration as $\langle u, q, v \rangle$.
– $\langle \varepsilon, q_0, w \rangle$ is the start configuration
– $\langle u, q_{acc}, w \rangle$ is the accepting configuration, and
– $\langle u, q_{rej}, w \rangle$ is the rejecting configuration.
– If $\delta(q_i, b) = (q_j, c, R)$ then $\langle ua, q_i, bv \rangle$ yields $\langle uac, q_j, v \rangle$, and
– If $\delta(q_i, b) = (q_j, c, L)$ then $\langle ua, q_i, bv \rangle$ yields $\langle u, q_j, acv \rangle$.
– A TM accepts an input string $w \in \Sigma^*$ if there is sequence of configurations $C_1, C_2, \ldots, C_n$ where C_1 is the initial configuration on w, C_n is an accepting configuration, and each C_i yields C_{i+1}.

Turing-Recognizability and Turing-Decidability

– A language L is called Turing-recognizable, or recursively-enumerable, if there is some Turing machine that recognizes it.

Turing-Recognizability and Turing-Decidability

- A language L is called Turing-recognizable, or recursively-enumerable, if there is some Turing machine that recognizes it.
- A language L is called co-recursively-enumerable (co-re) if its complement is Turing-recognizable.
- One every word a Turing machine may either accept, reject, or loop forever.

Turing-Recognizability and Turing-Decidability

- A language L is called Turing-recognizable, or recursively-enumerable, if there is some Turing machine that recognizes it.
- A language L is called co-recursively-enumerable (co-re) if its complement is Turing-recognizable.
- One every word a Turing machine may either accept, reject, or loop forever.
- We call a Turing machine that always make a decision to accept or reject on every input (never loops), is called a decider.
- A language L is called Turing decidable, or recursive, if there is some Turing machine that decided it.

Programming with Turing Machines

1. $L = \{0^{2^n} : n \geq 0\}$

Programming with Turing Machines

1. $L = \{0^{2^n} \ : \ n \geq 0\}$
2. TMs as transformers of tape $ww \mapsto \vdash w\#w \dashv$
3. L is a given regular language, say $L = \{(a+b)^* b (a+b)^*\}$.

Programming with Turing Machines

1. $L = \{0^{2^n} \; : \; n \geq 0\}$
2. TMs as transformers of tape $ww \mapsto \vdash w\#w \dashv$
3. L is a given regular language, say $L = \{(a+b)^* b (a+b)^*\}$.

   Tip 1. Simulate an DFA using a TM?
4. L is a given context-free language, say $L = \{w \; : \; w \text{ is a palindrome}\}$

Programming with Turing Machines

1. $L = \{0^{2^n} \; : \; n \geq 0\}$
2. TMs as transformers of tape $ww \mapsto \vdash w\#w \dashv$
3. L is a given regular language, say $L = \{(a+b)^* b (a+b)^*\}$.

   Tip 1. Simulate an DFA using a TM?
4. L is a given context-free language, say $L = \{w \; : \; w \text{ is a palindrome}\}$

   Tip 2. Storage in states

Programming with Turing Machines

1. $L = \{0^{2^n} \,:\, n \geq 0\}$
2. TMs as transformers of tape $ww \mapsto \vdash w\#w \dashv$
3. L is a given regular language, say $L = \{(a+b)^* b (a+b)^*\}$.

   Tip 1. Simulate an DFA using a TM?
4. L is a given context-free language, say $L = \{w \,:\, w \text{ is a palindrome}\}$

   Tip 2. Storage in states

   Tip 3. Simulate an PDA using a TM?

Programming with Turing Machines

1. $L = \{0^{2^n} \,:\, n \geq 0\}$
2. TMs as transformers of tape $ww \mapsto \vdash w\#w \dashv$
3. L is a given regular language, say $L = \{(a+b)^* b (a+b)^*\}$.
   Tip 1. Simulate an DFA using a TM?
4. L is a given context-free language, say $L = \{w \,:\, w \text{ is a palindrome}\}$
   Tip 2. Storage in states
   Tip 3. Simulate an PDA using a TM?
5. $L = \{a^n b^n c^n \,:\, n \geq 0\}$.

Programming with Turing Machines

1. $L = \{0^{2^n} : n \geq 0\}$
2. TMs as transformers of tape $ww \mapsto \vdash w\#w \dashv$
3. L is a given regular language, say $L = \{(a+b)^* b (a+b)^*\}$.

   Tip 1. Simulate an DFA using a TM?
4. L is a given context-free language, say $L = \{w : w$ is a palindrome$\}$

   Tip 2. Storage in states

   Tip 3. Simulate an PDA using a TM?
5. $L = \{a^n b^n c^n : n \geq 0\}$.

   Tip 5. Concepts of subroutines: CheckRegular($a^* b^* c^*$)

Programming with Turing Machines

1. $L = \{0^{2^n} : n \geq 0\}$
2. TMs as transformers of tape $ww \mapsto \vdash w\#w \dashv$
3. L is a given regular language, say $L = \{(a+b)^* b (a+b)^*\}$.

   Tip 1. Simulate an DFA using a TM?
4. L is a given context-free language, say $L = \{w : w \text{ is a palindrome}\}$

   Tip 2. Storage in states

   Tip 3. Simulate an PDA using a TM?
5. $L = \{a^n b^n c^n : n \geq 0\}$.

   Tip 5. Concepts of subroutines: CheckRegular$(a^* b^* c^*)$
6. $L = \{a^p : p \text{ is a prime number}\}$

Programming with Turing Machines

1. $L = \{0^{2^n} \ : \ n \geq 0\}$
2. TMs as transformers of tape $ww \mapsto \vdash w\#w \dashv$
3. L is a given regular language, say $L = \{(a+b)^*b(a+b)^*\}$.

   Tip 1. Simulate an DFA using a TM?
4. L is a given context-free language, say $L = \{w \ : \ w \text{ is a palindrome}\}$

   Tip 2. Storage in states

   Tip 3. Simulate an PDA using a TM?
5. $L = \{a^n b^n c^n \ : \ n \geq 0\}$.

   Tip 5. Concepts of subroutines: CheckRegular$(a^*b^*c^*)$
6. $L = \{a^p \ : \ p \text{ is a prime number}\}$

   Hint: Sieve of Eratosthenes

   Tip 4. Marking the tape/ Multiple Tracks
7. $L = \{a^i b^j c^k \ : \ i \times j = k \text{ and } i, j, k \geq 1\}$

Programming with Turing Machines

1. $L = \{0^{2^n} \ : \ n \geq 0\}$
2. TMs as transformers of tape $ww \mapsto \vdash w\#w \dashv$
3. L is a given regular language, say $L = \{(a+b)^* b(a+b)^*\}$.
   Tip 1. Simulate an DFA using a TM?
4. L is a given context-free language, say $L = \{w \ : \ w$ is a palindrome$\}$
   Tip 2. Storage in states
   Tip 3. Simulate an PDA using a TM?
5. $L = \{a^n b^n c^n \ : \ n \geq 0\}$.
   Tip 5. Concepts of subroutines: CheckRegular$(a^* b^* c^*)$
6. $L = \{a^p \ : \ p$ is a prime number$\}$
   Hint: Sieve of Eratosthenes
   Tip 4. Marking the tape/ Multiple Tracks
7. $L = \{a^i b^j c^k \ : \ i \times j = k$ and $i, j, k \geq 1\}$
   Check$(a^* b^* c^*)$, Delete(c, b)
8. $L = \{ww \ : \ w \in \{0,1\}^*\}$.
9. $L = \{x_1 \# x_2 \# \dots \# x_n \ : \ x_i \in \{0,1\}^*$ and $x_i \neq x_j$ for $i \neq j\}$.

TM for $L = \{a^p \ : \ p$ is a prime number$\}$

Algorithm 1.

1. If $p = 0$ or $p = 1$ reject.
2. Otherwise, Place a left end-marker $\vdash$, and erase the first a, and scan right to the end of the input and replace the last a with a \$.
3. Repeat:
   3.1 From the left endmarker, scan right and find the first non-blank cell, if it is at m position then m is a prime number. If this position is \$ then accept.
   3.2 Otherwise Mark this symbol with a $\star$ and all symbols before it till the left-endmarker with a prime $'$.
   3.3 Now we go to an inner loop erasing all a's that are at positions multiple of m. Repeat:
      3.3.1 Shift all the marks one cell at a time, finally moving the mark $\star$.
      3.3.2 Erase the symbol with the new $\star$ mark.
      3.3.3 If the new position with $\star$ mark is a \$ reject,
      3.3.4 If at anytime we visit a blank cell, exit this loop.
      3.3.5 Otherwise, go left to the first cell with prime $'$ mark, and repeat from 4.3.1.
   3.4 repeat from 4.1.

TM for $L = \{ww \; : \; w \in \{a, b\}^*\}$

Algorithm 2.

1. Place the endmarkers both sides of the tape, and reject if the input is of odd length.
2. Repeat
   2.1 Move to the left end-marker, and find the first unmarked symbol to the right, and replace it with its primed version. Exit the loop if there is no unmarked symbol to the right.
   2.2 Go to the last unmarked symbol in the right and replace it with its $\star$'d version.
3. Repeat
   3.1 Go to the leftmost primed symbol, erase it, remember it within the state, and go right to the first star'd symbol and match it with the just erased symbol stored in the state. If these two symbols are not the same reject, otherwise erase it, and goto 3.1.
   3.2 If there is no primed symbol, accept.

TM for $L = \{a^n b^n c^n \ : \ n \geq 0\}$

Algorithm 3.

1. Place the left and right endmarkers around the tape.
2. Check if the input is of the form $a^* b^* c^*$.
3. Repeat
   3.1 Go to the leftmost a. If there is no a, scan right for b or c. Accept in case there are no b's or c's. Reject otherwise.
   3.2 If there is a leftmost a, erase it, and go right to the leftmost b. If there is no b Reject, otherwise remove the b and scan right for a c.
   3.3 If there is no c Reject, otherwise erase the c, and goto 3.1.

Turing machines computing a partial function

- So far we have discussed TMs accepting a language.
- We can similarly define TMs to be computing partial functions, such that when a TM halts, the contents of the tape define the output of the function.
  - $w \mapsto \overline{w}$
  - $n \mapsto n \mod 2$
  - $n \mapsto n + 2$
  - $n \mapsto n^2$

Robustness of Turing Machines

The following extensions do not increase expressiveness of Turing machines.

1. Multi-tape Turing machines
2. Turing machines with Bi-infinite Tape
3. Nondeterministic Turing machines
4. Post machines or Queue automaton
5. PDAs with two stacks
6. Counter machines

Solving more challenging problems using TMs

1. Sorting a list $L = \{1^{n_1}01^{n_2}0\ldots1^{n_k} \;:\; n_1 \leq n_2 \leq \cdots \leq n_k\}$

Solving more challenging problems using TMs

1. Sorting a list $L = \{1^{n_1}01^{n_2}0\ldots1^{n_k} \;:\; n_1 \leq n_2 \leq \cdots \leq n_k\}$
2. Searching a list $L = \{1^n\#1^{n_1}01^{n_2}0\ldots1^{n_k} \;:\; n \in \{n_1, \ldots, n_k\}\}$

Solving more challenging problems using TMs

1. Sorting a list $L = \{1^{n_1}01^{n_2}0\dots1^{n_k} \ : \ n_1 \leq n_2 \leq \cdots \leq n_k\}$
2. Searching a list $L = \{1^n\#1^{n_1}01^{n_2}0\dots1^{n_k} \ : \ n \in \{n_1, \dots, n_k\}\}$
3. Substring matching
   $L = \{w\#w' \ : \ w \in \{a,b\}^*$ and w is a substring of $w'\}$.

Solving more challenging problems using TMs

1. Sorting a list $L = \{1^{n_1}01^{n_2}0\dots1^{n_k} \;:\; n_1 \leq n_2 \leq \cdots \leq n_k\}$
2. Searching a list $L = \{1^n\#1^{n_1}01^{n_2}0\dots1^{n_k} \;:\; n \in \{n_1,\dots,n_k\}\}$
3. Substring matching
   $L = \{w\#w' \;:\; w \in \{a,b\}^*$ and w is a substring of $w'\}$.
4. Subsequence search
5. Graph search

Solving more challenging problems using TMs

1. Sorting a list $L = \{1^{n_1}01^{n_2}0\dots1^{n_k} \ : \ n_1 \leq n_2 \leq \cdots \leq n_k\}$
2. Searching a list $L = \{1^n\#1^{n_1}01^{n_2}0\dots1^{n_k} \ : \ n \in \{n_1, \dots, n_k\}\}$
3. Substring matching
   $L = \{w\#w' \ : \ w \in \{a, b\}^*$ and w is a substring of $w'\}$.
4. Subsequence search
5. Graph search
6. Programmable Turing machine

Solving more challenging problems using TMs

1. Sorting a list $L = \{1^{n_1}01^{n_2}0\dots1^{n_k} \,:\, n_1 \leq n_2 \leq \cdots \leq n_k\}$
2. Searching a list $L = \{1^{n}\#1^{n_1}01^{n_2}0\dots1^{n_k} \,:\, n \in \{n_1, \dots, n_k\}\}$
3. Substring matching
   $L = \{w\#w' \,:\, w \in \{a,b\}^*$ and w is a substring of $w'\}$.
4. Subsequence search
5. Graph search
6. Programmable Turing machine aka Universal Turing machine

Turing Machines

Undecidability

Reductions

How to encode a TM in binary

Consider a TM $T = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, F)$ over the input alphabet $\{0, 1\}$.

1. Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ and $\Gamma = \{X_1, X_2, \ldots, X_m\}$.
2. Let's encode states and tape alphabet is unary as state q_i as string 0^i, and similarly tape symbol X_j as string 0^j.

How to encode a TM in binary

Consider a TM $T = (Q, \{0,1\}, \Gamma, \delta, q_1, B, F)$ over the input alphabet $\{0,1\}$.

1. Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ and $\Gamma = \{X_1, X_2, \ldots, X_m\}$.
2. Let's encode states and tape alphabet is unary as state q_i as string 0^i, and similarly tape symbol X_j as string 0^j.
3. Assume that 0^1 is start state, while 0^2 is the unique accept state.

How to encode a TM in binary

Consider a TM $T = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, F)$ over the input alphabet $\{0, 1\}$.

1. Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ and $\Gamma = \{X_1, X_2, \ldots, X_m\}$.
2. Let's encode states and tape alphabet is unary as state q_i as string 0^i, and similarly tape symbol X_j as string 0^j.
3. Assume that 0^1 is start state, while 0^2 is the unique accept state.
4. Assume that X_1 is 0, X_2 is 1, and X_3 is B.
5. We encode directions L and R as $D_1 = 0$ and $D_2 = 00$.

How to encode a TM in binary

Consider a TM $T = (Q, \{0,1\}, \Gamma, \delta, q_1, B, F)$ over the input alphabet $\{0,1\}$.

1. Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ and $\Gamma = \{X_1, X_2, \ldots, X_m\}$.
2. Let's encode states and tape alphabet is unary as state q_i as string 0^i, and similarly tape symbol X_j as string 0^j.
3. Assume that 0^1 is start state, while 0^2 is the unique accept state.
4. Assume that X_1 is 0, X_2 is 1, and X_3 is B.
5. We encode directions L and R as $D_1 = 0$ and $D_2 = 00$.
6. A transition τ given as $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$ can be encoded as $\sigma(\tau)$ given as
$$0^i 1 0^j 1 0^k 1 0^\ell 1 0^m$$
7. We can encode a TM with transitions $\tau_1, \tau_2, \ldots, \tau_n$ as binary string
$$\sigma(\tau_1) 11 \sigma(\tau_2) 11 \ldots \sigma(\tau_n)$$
8. Every binary string corresponds to at most one Turing machine, and all TMs corresponds to at least one binary string.

How to encode a TM in binary

Consider a TM $T = (Q, \{0,1\}, \Gamma, \delta, q_1, B, F)$ over the input alphabet $\{0,1\}$.

1. Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ and $\Gamma = \{X_1, X_2, \ldots, X_m\}$.
2. Let's encode states and tape alphabet is unary as state q_i as string 0^i, and similarly tape symbol X_j as string 0^j.
3. Assume that 0^1 is start state, while 0^2 is the unique accept state.
4. Assume that X_1 is 0, X_2 is 1, and X_3 is B.
5. We encode directions L and R as $D_1 = 0$ and $D_2 = 00$.
6. A transition τ given as $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$ can be encoded as $\sigma(\tau)$ given as
$$0^i 1 0^j 1 0^k 1 0^\ell 1 0^m$$
7. We can encode a TM with transitions $\tau_1, \tau_2, \ldots, \tau_n$ as binary string
$$\sigma(\tau_1) 11 \sigma(\tau_2) 11 \ldots \sigma(\tau_n)$$
8. Every binary string corresponds to at most one Turing machine, and all TMs corresponds to at least one binary string. garbage strings

How to encode a TM in binary

Consider a TM $T = (Q, \{0,1\}, \Gamma, \delta, q_1, B, F)$ over the input alphabet $\{0,1\}$.

1. Let $Q = \{Q_1, Q_2, \ldots, Q_n\}$ and $\Gamma = \{X_1, X_2, \ldots, X_m\}$.
2. Let's encode states and tape alphabet is unary as state q_i as string 0^i, and similarly tape symbol X_j as string 0^j.
3. Assume that 0^1 is start state, while 0^2 is the unique accept state.
4. Assume that X_1 is 0, X_2 is 1, and X_3 is B.
5. We encode directions L and R as $D_1 = 0$ and $D_2 = 00$.
6. A transition τ given as $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$ can be encoded as $\sigma(\tau)$ given as
$$0^i 1 0^j 1 0^k 1 0^\ell 1 0^m$$
7. We can encode a TM with transitions $\tau_1, \tau_2, \ldots, \tau_n$ as binary string
$$\sigma(\tau_1) 11 \sigma(\tau_2) 11 \ldots \sigma(\tau_n)$$
8. Every binary string corresponds to at most one Turing machine, and all TMs corresponds to at least one binary string. garbage strings
9. Hence, the set of possible TMs is countable.

The Language L_d

1. The set of binary strings is countable. Let's assign a unique integer to every binary string. We write w_i for the unique binary string corresponding to integer i.

The Language L_d

1. The set of binary strings is countable. Let's assign a unique integer to every binary string. We write w_i for the unique binary string corresponding to integer i. (How?)

The Language L_d

1. The set of binary strings is countable. Let's assign a unique integer to every binary string. We write w_i for the unique binary string corresponding to integer i. (How?)
2. We write M_i for the Turing machine corresponding to integer i.
3. Let L_d be the set of all strings w_i s.t. TM M_i does not accept w_i, i.e.

$$L_d = \{w_i \ : \ w_i \not\in L(M_i)\}.$$

Theorem

The language L_d is not recursively enumerable.

The Language L_d

1. The set of binary strings is countable. Let's assign a unique integer to every binary string. We write w_i for the unique binary string corresponding to integer i. (How?)
2. We write M_i for the Turing machine corresponding to integer i.
3. Let L_d be the set of all strings w_i s.t. TM M_i does not accept w_i, i.e.

$$L_d = \{w_i \,:\, w_i \not\in L(M_i)\}.$$

Theorem

The language L_d is not recursively enumerable.

Proof (via Diagonalization).

Assuming that there is a Turing machine M_d accepting L_d, i.e. $L_d = L(M_d)$ yields contradiction.

The Language L_d

1. The set of binary strings is countable. Let's assign a unique integer to every binary string. We write w_i for the unique binary string corresponding to integer i. (How?)
2. We write M_i for the Turing machine corresponding to integer i.
3. Let L_d be the set of all strings w_i s.t. TM M_i does not accept w_i, i.e.

$$L_d = \{w_i \;:\; w_i \notin L(M_i)\}.$$

Theorem

The language L_d is not recursively enumerable.

Proof (via Diagonalization).

Assuming that there is a Turing machine M_d accepting L_d, i.e. $L_d = L(M_d)$ yields contradiction.

1. If $w_d \in L(M_d)$ then $w_d \notin L_d$. Contradiction with $L_d = L(M_d)$.

The Language L_d

1. The set of binary strings is countable. Let's assign a unique integer to every binary string. We write w_i for the unique binary string corresponding to integer i. (How?)
2. We write M_i for the Turing machine corresponding to integer i.
3. Let L_d be the set of all strings w_i s.t. TM M_i does not accept w_i, i.e.

$$L_d = \{w_i \ : \ w_i \not\in L(M_i)\}.$$

Theorem

The language L_d is not recursively enumerable.

Proof (via Diagonalization).

Assuming that there is a Turing machine M_d accepting L_d, i.e. $L_d = L(M_d)$ yields contradiction.

1. If $w_d \in L(M_d)$ then $w_d \not\in L_d$. Contradiction with $L_d = L(M_d)$.
2. If $w_d \not\in L(M_d)$ then $w_d \in L_d$. Contradiction with $L_d = L(M_d)$.

□

An Undecidable language that is R.E.

- Recursive, Recursively Enumerable, and non-recursively-enumerable
- Decidable (recursive) and Undecidable (R.E. and non-R.E.).
- L_d is non-R.E.
- Can we find a language that is R.E. but undecidable (non-recursive)?

An Undecidable language that is R.E.

- Recursive, Recursively Enumerable, and non-recursively-enumerable
- Decidable (recursive) and Undecidable (R.E. and non-R.E.).
- L_d is non-R.E.
- Can we find a language that is R.E. but undecidable (non-recursive)?

Yes we can.

Halting Problem

Consider the language $L_U = \{0_i 111w \ : \ \text{TM } M_i \text{ accepts (halts on) input } w\}$

Theorem

The language L_U is recursively enumerable (i.e. there is a Turing machine, called Universal Turing machine, that accepts L_U).

Proof.

1. Turing machine uses four tapes—first to remember its input containing TM M_i and input w, second to simulate the tape of the TM M_i, the third to remember the current state of M_i, and fourth for additional work.
2. Such a TM accepts an input $0^i 111w$ iff TM M_i halts on the input w.

□

Undecidability of the Halting Problem L_U

Theorem

L_U is recursively enumerable but not recursive.

Proof.

1. We have already shown that L_U is recursively enumerable.
2. We will prove by contradiction that L_U is not recursive.
3. Assume that L_U is recursive, i.e. there exists a TM M_U to accept L_U that always halts.
4. We can then use this TM M_U to give a TM for L_d (details on the board), a contradiction.
5. Hence L_U is not recursive.

□

Turing Machines

Undecidability

Reductions

Programming Exercise

String Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Programming Exercise

String Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Example

Consider the lists
$A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$.

Programming Exercise

String Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Example

Consider the lists
$A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$.
There is a sequence $i = 2, 3, 1$ such that $s_2 s_3 s_1 = t_2 t_3 t_1$, since

- $s_2 s_3 s_1 = 00110110110$ and $t_2 t_3 t_1 = 00110110110$.

Programming Exercise

String Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Example

Consider the lists
$A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$.
There is a sequence $i = 2, 3, 1$ such that $s_2 s_3 s_1 = t_2 t_3 t_1$, since

- $s_2 s_3 s_1 = 00110110110$ and $t_2 t_3 t_1 = 00110110110$.

Interesting cases

1. Consider $A = \langle 0011, 11, 1101 \rangle$ and $B = \langle 101, 011, 110 \rangle$.

Programming Exercise

String Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n\rangle$ and $B = \langle t_1, t_2, \ldots, t_n\rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Example

Consider the lists
$A = \langle 110, 0011, 0110\rangle$ and $B = \langle 110110, 00, 110\rangle$.
There is a sequence $i = 2, 3, 1$ such that $s_2 s_3 s_1 = t_2 t_3 t_1$, since

- $s_2 s_3 s_1 = 00110110110$ and $t_2 t_3 t_1 = 00110110110$.

Interesting cases

1. Consider $A = \langle 0011, 11, 1101\rangle$ and $B = \langle 101, 011, 110\rangle$. (no solution)

Programming Exercise

String Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Example

Consider the lists
$A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$.
There is a sequence $i = 2, 3, 1$ such that $s_2 s_3 s_1 = t_2 t_3 t_1$, since

- $s_2 s_3 s_1 = 00110110110$ and $t_2 t_3 t_1 = 00110110110$.

Interesting cases

1. Consider $A = \langle 0011, 11, 1101 \rangle$ and $B = \langle 101, 011, 110 \rangle$. (no solution)
2. Consider $A = \langle 100, 0, 1 \rangle$ and $B = \langle 1, 100, 0 \rangle$.

Programming Exercise

String Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \dots, s_n \rangle$ and $B = \langle t_1, t_2, \dots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \dots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \dots s_{i_n} = t_{i_1} t_{i_2} \dots t_{i_n}.$$

Example

Consider the lists
$A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$.
There is a sequence $i = 2, 3, 1$ such that $s_2 s_3 s_1 = t_2 t_3 t_1$, since

- $s_2 s_3 s_1 = 00110110110$ and $t_2 t_3 t_1 = 00110110110$.

Interesting cases

1. Consider $A = \langle 0011, 11, 1101 \rangle$ and $B = \langle 101, 011, 110 \rangle$. (no solution)
2. Consider $A = \langle 100, 0, 1 \rangle$ and $B = \langle 1, 100, 0 \rangle$. (shortest sol. len. 75)!!!

Programming Exercise

String-List Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Can you design an algorithm to solve this problem?

Programming Exercise

String-List Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Can you design an algorithm to solve this problem? A semi-algorithm?

Programming Exercise

String-List Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Can you design an algorithm to solve this problem? A semi-algorithm?

Theorem

There is no algorithm for the string-list matching problem (also known as Post's correspondence problem (PCP)).

Programming Exercise

String-List Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Can you design an algorithm to solve this problem? A semi-algorithm?

Theorem

There is no algorithm for the string-list matching problem (also known as Post's correspondence problem (PCP)). In other words, this problem is undecidable.

Programming Exercise

String-List Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Can you design an algorithm to solve this problem? A semi-algorithm?

Theorem

There is no algorithm for the string-list matching problem (also known as Post's correspondence problem (PCP)). In other words, this problem is undecidable.

But how do you prove it?

Programming Exercise

String-List Matching Problem $MATCH(A, B)$

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, whether there exists a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}.$$

Can you design an algorithm to solve this problem? A semi-algorithm?

Theorem

There is no algorithm for the string-list matching problem (also known as Post's correspondence problem (PCP)). In other words, this problem is undecidable.

But how do you prove it?

Q: Is PCP recursively-enumerable?

Reductions

Definition (Problem Reduction)

A reduction from problem P_1 to problem P_2 is an algorithm to convert instances of a problem P_1 to instances of problem P_2 that have same answers.
In this case we say that P_2 is as hard as P_1.

Reductions

Definition (Problem Reduction)

A reduction from problem P_1 to problem P_2 is an algorithm to convert instances of a problem P_1 to instances of problem P_2 that have same answers.
In this case we say that P_2 is as hard as P_1.

Theorem

If there is a reduction from problem P_1 to problem P_2, then

1. *If P_1 is undecidable then so is P_2.*
2. *If P_1 is non-RE then so is P_2.*

Reductions

Definition (Problem Reduction)

A reduction from problem P_1 to problem P_2 is an algorithm to convert instances of a problem P_1 to instances of problem P_2 that have same answers.
In this case we say that P_2 is as hard as P_1.

Theorem

If there is a reduction from problem P_1 to problem P_2, then

1. *If P_1 is undecidable then so is P_2.*
2. *If P_1 is non-RE then so is P_2.*

Proof by contradiction.

Recap

- Recall the languages (problems) L_d (Diagonal language) and L_U (Universal language).
- L_d is the set of TMs that do not accept (halt on) themselves.
- L_U is the set of pairs (M, w) such that TM M halts on w.

Recap

- Recall the languages (problems) L_d (Diagonal language) and L_U (Universal language).
- L_d is the set of TMs that do not accept (halt on) themselves.
- L_U is the set of pairs (M, w) such that TM M halts on w.
- L_d is non-RE and L_U is RE but not recursive.
- We can use a reduction from L_d and L_U to prove a problem non-RE and undecidable.

Some Reduction Based Proofs

Theorem

If L is recursive then so is the complement of L.

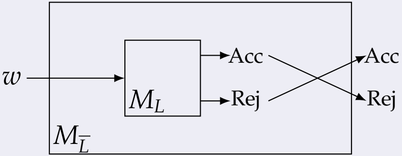
Proof.

Some Reduction Based Proofs

Theorem

If L is recursive then so is the complement of L.

Proof.



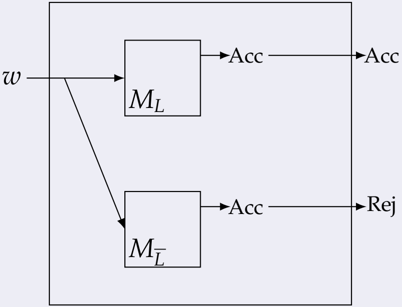
∎

Some Reduction Based Proofs

Theorem

If both L and complement of L are RE, then L is recursive.

Proof.



∎

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts some string, i.e. $L(M_i) \neq \emptyset\}$.

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts some string, i.e. $L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable.

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts some string, i.e. $L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable. Show a TM!
  - NE_{TM} is not recursive.

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts some string, i.e. $L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable. Show a TM!
  - NE_{TM} is not recursive. Show a reduction from L_U.

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts some string, i.e. $L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable. Show a TM!
  - NE_{TM} is not recursive. Show a reduction from L_U.
- E_{TM}, the complement of NE_{TM}.

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts some string, i.e. $L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable. Show a TM!
  - NE_{TM} is not recursive. Show a reduction from L_U.
- E_{TM}, the complement of NE_{TM}. not RE!

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts some string, i.e. $L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable. Show a TM!
  - NE_{TM} is not recursive. Show a reduction from L_U.
- E_{TM}, the complement of NE_{TM}. not RE!
- $ACC01_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts string 01, i.e. $01 \in L(M_i)\}$.

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \; : \; M_i$ accepts some string, i.e. $L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable. Show a TM!
  - NE_{TM} is not recursive. Show a reduction from L_U.
- E_{TM}, the complement of NE_{TM}. not RE!
- $ACC01_{TM} = \{\langle M_i \rangle \; : \; M_i$ accepts string 01, i.e. $01 \in L(M_i)\}$.
- $REG_{TM} = \{\langle M_i \rangle \; : \; M_i$ accepts a regular language$\}$.

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \;:\; M_i \text{ accepts some string, i.e. } L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable. Show a TM!
  - NE_{TM} is not recursive. Show a reduction from L_U.
- E_{TM}, the complement of NE_{TM}. not RE!
- $ACC01_{TM} = \{\langle M_i \rangle \;:\; M_i \text{ accepts string 01, i.e. } 01 \in L(M_i)\}$.
- $REG_{TM} = \{\langle M_i \rangle \;:\; M_i \text{ accepts a regular language}\}$.
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \;:\; L(M_1) = L(M_2)\}$.

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts some string, i.e. $L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable. Show a TM!
  - NE_{TM} is not recursive. Show a reduction from L_U.
- E_{TM}, the complement of NE_{TM}. not RE!
- $ACC01_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts string 01, i.e. $01 \in L(M_i)\}$.
- $REG_{TM} = \{\langle M_i \rangle \ : \ M_i$ accepts a regular language$\}$.
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \ : \ L(M_1) = L(M_2)\}$.

Theorem (Rice's Theorem)

Every nontrivial property of the RE languages in undecidable.

Undecidable Problems

Decide whether the following problems are recursive, RE, non-RE:

- $NE_{TM} = \{\langle M_i \rangle \;:\; M_i \text{ accepts some string, i.e. } L(M_i) \neq \emptyset\}$.
  - NE_{TM} is recursively-enumerable. Show a TM!
  - NE_{TM} is not recursive. Show a reduction from L_U.
- E_{TM}, the complement of NE_{TM}. not RE!
- $ACC01_{TM} = \{\langle M_i \rangle \;:\; M_i \text{ accepts string 01, i.e. } 01 \in L(M_i)\}$.
- $REG_{TM} = \{\langle M_i \rangle \;:\; M_i \text{ accepts a regular language}\}$.
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \;:\; L(M_1) = L(M_2)\}$.

Theorem (Rice's Theorem)

Every nontrivial property of the RE languages in undecidable.

Proof of Theorem 9.11 from HMU.

Post's Correspondence Problem

Theorem

Post's Correspondence Problem is undecidable.

Proof.

1. Reduction from the halting problem L_U instances (M, w) to a PCP instances s, t.
2. We encode a computation $\#\alpha_1\#\alpha_2\# \ldots \#$... where α_1 is the initial configuration of M on w, and each α_i and α_{i+1} is a valid transition of M, such that
   - Partial solutions of PCP problem will consists of prefixes of the unique computation of M on W
   - Solutions form t list will always be one configuration ahead than list s, unless M enters an accepting state, and then s list will be permitted to catch up with the t list and eventually produce a solution.
   - However, if the computation does not encounter an accepting state, the two partial solutions will never match, and hence no solution exists.

$\square$

Reduction Sketch

1. Modified Post's Correspondence Problem
2. The first pair is

List s	List t
$\#$	$\#q_0w$

3. Tape symbols $X \in \Gamma$ and separator $\#$ can be appended to both lists:

List s	List t	
X	X	for every $X \in \Gamma$
$\#$	$\#$	

4. Simulate one move of M, for all non accepting states

List s	List t	
qX	Yp	if $\delta(q, X) = (p, Y, R)$
ZqX	pZY	if $\delta(q, X) = (p, Y, L)$
$q\#$	$Yp\#$	if $\delta(q, B) = (p, Y, R)$
$Zq\#$	$pZY\#$	if $\delta(q, B) = (p, Y, L)$.

Reduction Sketch: Contd

5 For the accepting state

List s	List t
XqY	q
Xq	q
qY	q.

5 Once all the tape symbols have been consumed, we use the final pair

List s	List t
$q\#\#$	$\#$

to complete the solution.

Applications of PCPs

Theorem

Deciding ambiguity of CFGs is undecidable.

Proof.

Let $MATCH(A, B)$ be a PCP instance where $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$. Consider the CFG

$$\begin{array}{rcl} S & \rightarrow & A \mid B \\ A & \rightarrow & s_i A a_i \mid s_i a_i \\ B & \rightarrow & t_i B a_i \mid t_i a_i. \end{array}$$

It is easy to see that the grammar is ambiguous iff there the corresponding PCP has a solution. $\square$