CS310 : Automata Theory 2019

Lecture 24: Turing Machines and Computability

Instructor: S. Akshay

IITB, India

05-04-2019

Turing Machines

Definition

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

1. Q is a finite set of states,
2. Σ is a finite input alphabet,
3. Γ is a finite tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. q_0 is the start state,
5. q_{acc} is the accept state,
6. q_{rej} is the reject state,
7. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function.

Turing Machines

Definition

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

1. Q is a finite set of states,
2. Σ is a finite input alphabet,
3. Γ is a finite tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. q_0 is the start state,
5. q_{acc} is the accept state,
6. q_{rej} is the reject state,
7. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function.

For a $q \in Q$, $a \in \Gamma$ if $\delta(q, a) = (q', b, L)$, then

- q' is the new state of the machine,
- b is the letter which replaces a on the tape,
- the head moves to Left of the current position.

Configurations

A *configuration* is a snapshot of the system during computation.
Described by:

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Notation: $C = uqv$

where, uv is the tape content, current state is q and head is at start of v

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Notation: $C = uqv$

where, uv is the tape content, current state is q and head is at start of v

We define C_1 *yields* C_2 if the TM can move from C_1 to C_2 in one step:

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Notation: $C = uqv$

where, uv is the tape content, current state is q and head is at start of v

We define C_1 *yields* C_2 if the TM can move from C_1 to C_2 in one step:

- left move: $u\ a\ q_i\ b\ v$ yields $u\ q_j\ a\ c\ v$ if $\delta(q_i, b) = (q_j, c, L)$
- right move: $u\ a\ q_i\ b\ v$ yields $u\ a\ c\ q_j\ v$ if $\delta(q_i, b) = (q_j, c, R)$

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Notation: $C = uqv$

where, uv is the tape content, current state is q and head is at start of v

We define C_1 *yields* C_2 if the TM can move from C_1 to C_2 in one step:

- left move: $u\ a\ q_i\ b\ v$ yields $u\ q_j\ a\ c\ v$ if $\delta(q_i, b) = (q_j, c, L)$
- right move: $u\ a\ q_i\ b\ v$ yields $u\ a\ c\ q_j\ v$ if $\delta(q_i, b) = (q_j, c, R)$
- left-end: $q_i\ b\ v$ yields (1) $q_j\ c\ v$ if transition is left moving or (2) $c\ q_j\ v$ if it is right moving
- right-end: assume that $u\ a\ q_i$ is the same as $u\ a\ q_i \sqcup$ as tape has blanks to the right.

Computation of a Turing Machine

- We define start (q_0, w), accepting $(*q_{acc}*)$, rejecting $(*q_{rej}*)$ and halting configurations.

Computation of a Turing Machine

- We define start (q_0, w), accepting $(*q_{acc}*)$, rejecting $(*q_{rej}*)$ and halting configurations.
- A Turing Machine computation on a given input may not halt!

Computation of a Turing Machine

- We define start (q_0, w), accepting $(*q_{acc}*)$, rejecting $(*q_{rej}*)$ and halting configurations.

- A Turing Machine computation on a given input may not halt!

- A TM M *accepts* input word w if there exists a sequence of configurations $C_1, C_2, \ldots, C_k$ (called a run) such that
  - C_1 is the start configuration
  - each C_i yields C_{i+1}
  - C_k is an accepting configuration

Computation of a Turing Machine

- We define start (q_0, w), accepting $(*q_{acc}*)$, rejecting $(*q_{rej}*)$ and halting configurations.
- A Turing Machine computation on a given input may not halt!
- A TM M *accepts* input word w if there exists a sequence of configurations $C_1, C_2, \ldots, C_k$ (called a run) such that
  - C_1 is the start configuration
  - each C_i yields C_{i+1}
  - C_k is an accepting configuration
- Language of TM M, denoted $L(M)$, is the set of strings accepted by it.

Turing recognizable and decidable languages

Turing recognizable or Recursively enumerable (r.e)

A language is Turing recognizable or r.e if there is a Turing machine accepting it.

Turing recognizable and decidable languages

Turing recognizable or Recursively enumerable (r.e)

A language is Turing recognizable or r.e if there is a Turing machine accepting it.

Turing decidable or recursive

A language is decidable (or recursive) if there is a Turing machine accepting it, which has the additional property that it halts on all possible inputs.

Turing recognizable and decidable languages

Turing recognizable or Recursively enumerable (r.e)

A language is Turing recognizable or r.e if there is a Turing machine accepting it.

Turing decidable or recursive

A language is decidable (or recursive) if there is a Turing machine accepting it, which has the additional property that it halts on all possible inputs.

Every decidable language is Turing recognizable, but is the converse true?

Another example of a Turing Machine

A TM as a computer of integer functions

- Compute $\max\{m-n, 0\}$: construct TM M that starts with 0^m10^n and halts with $0^{\max\{m-n,0\}}$ on tape.

Another example of a Turing Machine

A TM as a computer of integer functions

- Compute $\max\{m-n, 0\}$: construct TM M that starts with 0^m10^n and halts with $0^{\max\{m-n,0\}}$ on tape.

Same as defining $B = \{a^m b^n c^{\max\{m-n,0\}} \mid m, n \geq 0\}$ and asked for a TM which accepts language B.

Another example of a Turing Machine

A TM as a computer of integer functions

- Compute $\max\{m-n, 0\}$: construct TM M that starts with 0^m10^n and halts with $0^{\max\{m-n,0\}}$ on tape.

- M repeatedly replaces leading 0 by blank, then searches right for a 1 followed by 0 and changes 0 to 1.
- Then, M moves left until it encounters a blank and repeats this cycle.

Another example of a Turing Machine

A TM as a computer of integer functions

- Compute $\max\{m-n, 0\}$: construct TM M that starts with 0^m10^n and halts with $0^{\max\{m-n,0\}}$ on tape.

- M repeatedly replaces leading 0 by blank, then searches right for a 1 followed by 0 and changes 0 to 1.
- Then, M moves left until it encounters a blank and repeats this cycle.
- The repetition ends if
  1. searching right for 0, M encounters a blank.
     - Then all n 0's in 0^m10^n have been changed to 1 and $n+1$ of m 0's changed to blank. M replaces $n+1$ 1's by a 0 and n blanks leaving $m-n$ 0's on the tape.

Another example of a Turing Machine

A TM as a computer of integer functions

- Compute $\max\{m-n,0\}$: construct TM M that starts with 0^m10^n and halts with $0^{\max\{m-n,0\}}$ on tape.

- M repeatedly replaces leading 0 by blank, then searches right for a 1 followed by 0 and changes 0 to 1.
- Then, M moves left until it encounters a blank and repeats this cycle.
- The repetition ends if
  1. searching right for 0, M encounters a blank.
     - Then all n 0's in 0^m10^n have been changed to 1 and $n+1$ of m 0's changed to blank. M replaces $n+1$ 1's by a 0 and n blanks leaving $m-n$ 0's on the tape.
  2. beginning the cycle, M cannot find a 0 to change to a blank, because first m 0's have already been changed.
     - Then $n \geq m$, so $\max\{m-n,0\} = 0$. M replaces remaining 1's, 0's by blanks.

More examples/exercises

Exercises

1. $C = \{0^{2^n} \mid n \geq 0\}$, i.e., all strings of 0's whose length is a power of 2.
   - Give a high level description of a TM that accepts C.
   - Give the formal TM construction and state-diagram.

More examples/exercises

Exercises

1. $C = \{0^{2^n} \mid n \geq 0\}$, i.e., all strings of 0's whose length is a power of 2.
   - Give a high level description of a TM that accepts C.
   - Give the formal TM construction and state-diagram.
2. $D = \{a^i b^j c^k \mid i * j = k \text{ and } i, j, k \geq 1\}$. Give a (high level description of) a TM that accepts D.

Why Turing Machines?

- Robust model of computation
- Equivalence with other such models of computation, with reasonable assumptions (e.g., only finite amount of work is possible in 1 step).

Why Turing Machines?

- Robust model of computation
- Equivalence with other such models of computation, with reasonable assumptions (e.g., only finite amount of work is possible in 1 step).
- Thus, though there are several computational models, the class of algorithms (or procedures) they describe is the same.

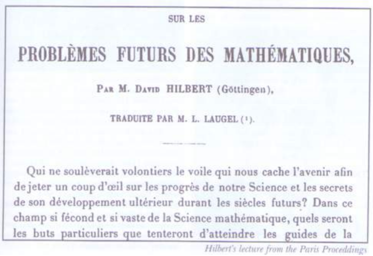
Why Turing Machines?

- Robust model of computation
- Equivalence with other such models of computation, with reasonable assumptions (e.g., only finite amount of work is possible in 1 step).
- Thus, though there are several computational models, the class of algorithms (or procedures) they describe is the same.
- Can do everything a computer can do and vice versa. But takes a lot more time. Is not practical and indeed its not what is implemented in today's computer. After all who wants to write 100 lines to do subtraction or check something that a 4-year old can do?

Why Turing Machines?

- Robust model of computation
- Equivalence with other such models of computation, with reasonable assumptions (e.g., only finite amount of work is possible in 1 step).
- Thus, though there are several computational models, the class of algorithms (or procedures) they describe is the same.
- Can do everything a computer can do and vice versa. But takes a lot more time. Is not practical and indeed its not what is implemented in today's computer. After all who wants to write 100 lines to do subtraction or check something that a 4-year old can do?
- So then again, why Turing? Why is the top-most nobel-equivalent award in computer science called Turing award and not Bill Gates award?

Turing Machines: a lesson from history



SUR LES

PROBLÈMES FUTURS DES MATHÉMATIQUES,

PAR M. DAVID HILBERT (Göttingen),

TRADUITE PAR M. L. LAUGEL (¹).

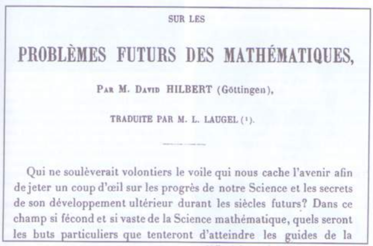
Qui ne soulèverait volontiers le voile qui nous cache l'avenir afin de jeter un coup d'œil sur les progrès de notre Science et les secrets de son développement ultérieur durant les siècles futurs? Dans ce champ si fécond et si vaste de la Science mathématique, quels seront les buts particuliers que tenteront d'atteindre les guides de la

Hilbert's lecture from the Paris Proceedings

Hilbert's problems

- In 1900, David Hilbert listed out 23 problems as challenges for 20th century at the Int. Cong. of Mathematicians in Paris.
- 10th problem: Devise an algorithm (a process doable using a finite no. of operations) to test if a given polynomial has integral roots.

Turing Machines: a lesson from history



SUR LES

PROBLÈMES FUTURS DES MATHÉMATIQUES,

PAR M. DAVID HILBERT (Göttingen),

TRADUITE PAR M. L. LAUGEL (¹).

Qui ne soulèverait volontiers le voile qui nous cache l'avenir afin de jeter un coup d'œil sur les progrès de notre Science et les secrets de son développement ultérieur durant les siècles futurs? Dans ce champ si fécond et si vaste de la Science mathématique, quels seront les buts particuliers que tenteront d'atteindre les guides de la

Hilbert's lecture from the Paris Proceedings

Hilbert's problems

- In 1900, David Hilbert listed out 23 problems as challenges for 20th century at the Int. Cong. of Mathematicians in Paris.
- 10th problem: Devise an algorithm (a process doable using a finite no. of operations) to test if a given polynomial has integral roots.
- Now we know that no such algorithm exists. But how to prove this without a mathematical definition of an algorithm?

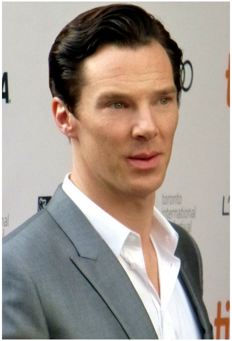
The Entscheidungsproblem

A challenge... in 1928

Is there an algorithm to decide whether any given statement is provable from the axioms using the rules of logic?

The Church-Turing Thesis

Alonso Church
(1903–1995)



Alan Turing
(1912–1954)

The Church-Turing Thesis

Alonso Church
(1903–1995)

Alan Turing
(1912–1954)

The Church-Turing Thesis

- The Church-Turing Thesis provides the definition of algorithm necessary to resolve Hilbert's tenth problem.

The Church-Turing Thesis

Our intuitive understanding of algorithms coincides with Turing machine algorithms.

The Church-Turing Thesis

- The Church-Turing Thesis provides the definition of algorithm necessary to resolve Hilbert's tenth problem.

The Church-Turing Thesis

Our intuitive understanding of algorithms coincides with Turing machine algorithms.

- Thus, Turing machines capture our intuitive idea of computation.
- Indeed, we are more interested in algorithms themselves. Once we believe that TMs precisely capture algorithms, we will shift our focus to algorithms rather than low-level descriptions of TMs.

More on the Church-Turing Thesis

Lambda Calculus

- In 1936, Church introduced Lambda Calculus as a formal description of all computable functions.
- Independently, Turing had introduced his A-machines in 1936 too.
- Turing also showed that his A-machines were equivalent to Lambda Calculus of Church.

- So, can a Turing machine do everything? In other words are there algorithms to solve every question. Godel's incompleteness result asserts otherwise.
- If there is TM solving a problem, does there exist an equivalent TM that halts?

What else did Turing do?

- Alan Turing characterized computable functions by building a machine. Though theoretical this gave rise to the idea of computers.
- But Turing also worked on ideas and concepts that later made profound impact in AI and cryptography.

What else did Turing do?

- Alan Turing characterized computable functions by building a machine. Though theoretical this gave rise to the idea of computers.
- But Turing also worked on ideas and concepts that later made profound impact in AI and cryptography.
- "Father" of modern computers and computer science.