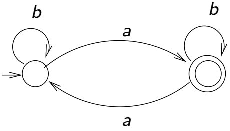
Overview of UMC 205: Automata and Computability

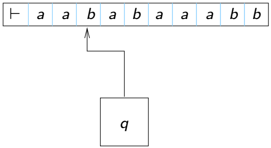
Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

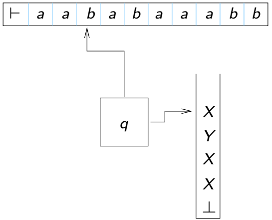
02 January 2024

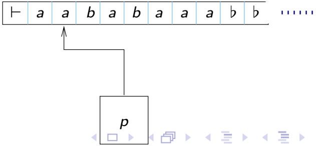
Different Kinds of “Automata” or “State Machines”



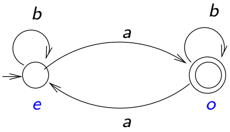


- Finite-State Automata
- Pushdown Automata
- Turing Machines

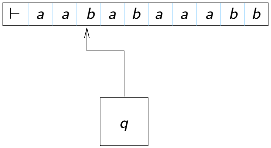


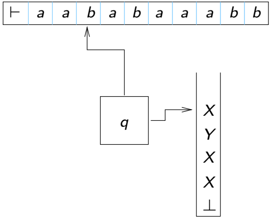


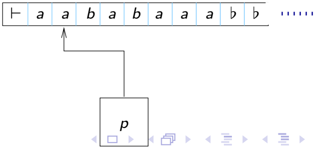
Different Kinds of “Automata” or “State Machines”



- Finite-State Automata
- Pushdown Automata
- Turing Machines







Kind of results we study in Automata Theory

- Expressive power of the models in terms of the class of languages they define.
  - Characterisations of this class of languages
    - Myhill-Nerode theorem.
    - Regular Expressions
    - Algebraic (in terms of monoids)
  - Necessary conditions these classes satisfy
    - Pumping Lemma and ultimate periodicity (for Regular/CFL).
    - Parikh's Theorem (for Context-Free Languages).
- Decision procedures
  - Emptiness problem
  - Language inclusion problem
  - Configuration reachability problem.
- Computability (Turing machines give a compelling notion of computable functions), Rice's Theorem.

Why study automata theory?

Corner stone of many subjects in CS:

1. Compilers
   - Lexical analysis, parsing, regular expression search
2. Digital circuits (state minimization, analysis).
3. Computability, Complexity Theory (algorithmic hardness of problems)
4. Mathematical Logic
   - Decision procedures for logical problems.
5. Formal Verification
   - Configuration reachability
   - Is $L(\mathcal{A}) \subseteq L(\mathcal{B})$?

Uses in Verification

1. System models are natural extensions of automata models
   - Programs with no dynamic memory allocation, no procedures = Finite State systems.
   - No dynamic memory allocation = Pushdown systems.
   - General program = Turing machine.
   - Programs with unbounded integer variables = Counter machines.

   Decision procedures for emptiness, configuration reachability, etc, directly translate to decision procedures for programs.
2. To solve "model-checking" problem for logics that talk about infinite behaviour.

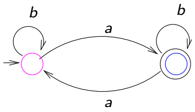
Uses in Logic

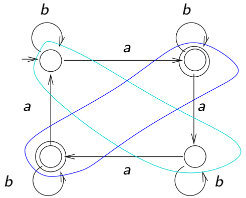
- Obtain decision procedure for satisfiability of a logic by translating a formula to an automaton and checking emptiness.
- Argue undecidability/incompleteness of a proof system.

Myhill-Nerode Theorem

Myhill-Nerode Theorem:

Every regular language has a canonical DFA accepting it.





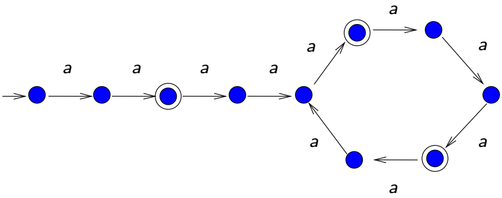
Some consequences:

- Any DFA for L is a *refinement* of its canonical DFA.
- "minimal" DFA's for L are isomorphic.

Ultimate Periodicity of Regular Languages

Claim

If L is a regular language then $lengths(L)$ is an ultimately periodic set.



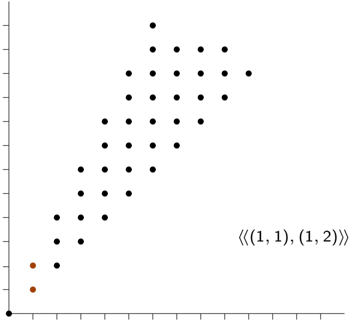
$$lengths(L(\mathcal{A})) = \{2\} \cup \{5, 11, 17, \ldots\} \cup \{8, 14, 20, \ldots\}.$$

Parikh's Theorem for CFL's

$\psi(w)$: "Letter-count" of a string w:

$$\text{Eg} : \psi(aabab) = (3, 2).$$

If L is a context-free language, then $\psi(L)$ is semi-linear (Every CFL is letter-equivalent to a regular language).



Can be used to show certain languages are *not* context-free: Eg. $L = \{a^{2^n} \mid n \geq 0\}$.

Computable functions

Consider functions of natural numbers

$$f : \mathbb{N} \to \mathbb{N}.$$

When do we say a function f is "computable"?

Computable functions

Consider functions of natural numbers

$$f : \mathbb{N} \to \mathbb{N}.$$

When do we say a function f is "computable"?
If we can give a "finite recipe" to compute the value of $f(n)$ for any input n.

Example: a recipe to compute the sum of two numbers

```
input: m, n
1. i := length(m);
2. ans := "";
3. add m[i] and n[i] to get sum s and carry c
4. ans := s :: ans;
5. if (i == 1)
     ans := c :: ans;
   else
     i := i-1;
     goto step 3;
6. Output ans
```

```
65932
15823
-----
81755
-----
```

More examples

$$f(n) = n \mapsto 2^n$$

$$flt(n) = \begin{cases} 1 & \text{if } \exists x, y, z: \; x^n + y^n = z^n \\ 0 & \text{otherwise} \end{cases}$$

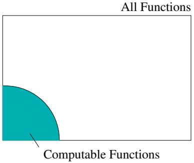
$$hp(n) = \begin{cases} 1 & \text{if } n \text{ encodes a halting Turing machine} \\ 0 & \text{otherwise} \end{cases}$$

Density of Computable Functions

Are there any uncomputable functions?

Density of Computable Functions

Are there any uncomputable functions?



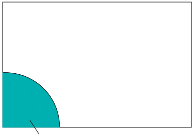
Uncountably infinitely many!

Use a diagonalization argument:

	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
f_0	0	0	0	0	0	0	0	0	0	0	0	0	$\cdots$
f_1	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
f_2	12	10	8	6	4	2	0	2	4	6	8	10	$\cdots$
f_3	69	0	1	0	42	7	0	0	0	8	0	9	$\cdots$
f_4	0	1	0	0	9	1	1	0	0	0	0	0	$\cdots$
f_5	1	11	0	10	0	1	1	5	0	1	61	1	$\cdots$
f_6	0	1	0	0	0	1	1	0	0	0	0	0	$\cdots$
f_7	0	0	13	0	6	0	1	15	0	0	1	0	$\cdots$
$\vdots$									$\ddots$				

Density of Computable Functions

Are there any uncomputable functions?

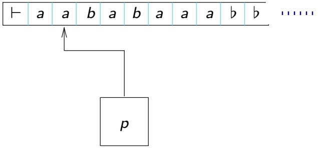


Uncountably infinitely many!

Use a diagonalization argument:

	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
f_0	0	0	0	0	0	0	0	0	0	0	0	0	$\cdots$
f_1	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
f_2	12	10	8	6	4	2	0	2	4	6	8	10	$\cdots$
f_3	69	0	1	0	42	7	0	0	0	8	0	9	$\cdots$
f_4	0	1	0	0	9	1	1	0	0	0	0	0	$\cdots$
f_5	1	11	0	10	0	1	1	5	0	1	61	1	$\cdots$
f_6	0	1	0	0	0	1	1	0	0	0	0	0	$\cdots$
f_7	0	0	13	0	6	0	1	15	0	0	1	0	$\cdots$
$\vdots$									$\ddots$				
g	1	3	9	1	10	2	2	16	0	0	0	0	$\cdots$

How a Turing machine works

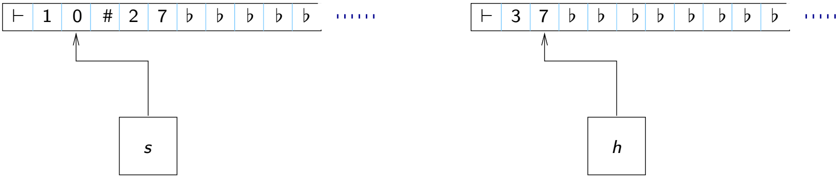


- Finite control
- Tape infinite to the right
- Each step: In current state p, read current symbol under the tape head, say a: Change state to q, replace current symbol by b, and move head left or right.

$$(p, a) \rightarrow (q, b, L/R).$$

Definition of computability

- A function $f : \mathbb{N} \to \mathbb{N}$ is computable if there is a Turing machine M which when started with n on its tape, always halts with $f(n)$ on its tape.



- Finite recipe = A Turing machine that always halts
- Can give a simple TM that computes the addition function.

Gödel's Incompleteness result

There cannot be a sound and complete proof system for first-order arithmetic.

What we can say in $\mathrm{FO}(\mathbb{N}, +, \cdot)$

- "Every number has a successor"

$$\forall n \exists m (m = n + 1).$$

- "Every number has a predecessor"

$$\forall n \exists m (n = m + 1).$$

- "There are only finitely many primes"

$$\exists n \forall p (prime(p) \implies p < n).$$

- "There are infinitely many primes"

$$\forall n \exists p (prime(p) \land p > n).$$

Peano's Proof System for Arithmetic

- Axioms:

$$
\begin{gathered}
\forall x \neg(0 = x+1) \\
\forall x \forall y (x+1 = y+1 \implies x = y) \\
\forall x (x+0 = x) \\
\forall x \forall y \forall z (x + (y+z) = (x+y)+z) \\
\forall x (x \cdot 0 = 0) \\
\forall x \forall y \forall z (x \cdot (y+z) = ((x \cdot y) + (x \cdot z))) \\
(\varphi(0) \ \& \ \forall x(\varphi(x) \implies \varphi(x+1))) \implies \forall x \varphi(x).
\end{gathered}
$$

- Other axioms like $(\varphi \ \& \ \psi) \implies \varphi$, $\forall x(\varphi) \implies \varphi(17)$.
- Inference rules like "Modus Ponens"

$$\text{Given } \varphi \text{ and } \varphi \implies \psi, \ \text{infer } \psi.$$

Proof

A proof of φ in a proof system is a finite sequence of sentences

$$\varphi_0, \varphi_1, \dots, \varphi_n$$

such that each φ_i is either an axiom or follows from two previous ones by an inference rule, and $\varphi_n = \varphi$.

Notion of $X \vdash_{\mathcal{G}} \varphi$.

A proof system is "sound" if whatever it proves is indeed true (i.e. in $Th(\mathbb{N}, +, \cdot)$).
A proof system is "complete" if it can prove whatever is true (i.e. in $Th(\mathbb{N}, +.\cdot)$).

Gödel's Incompleteness result

> *There cannot be a sound and complete proof system for first-order arithmetic.*

Formal language-theoretic proof: $\mathrm{Th}(\mathbb{N}, +, .)$ is not even recursively enumerable.

Course details

- Weightage: 40% assignments + quizes + seminar, 20% midsem exam, 40% final exam.
- Assignments to be done on your own.
- Dishonesty Policy: Any instance of copying in an assignment will fetch you a 0 in that assignment + one grade reduction.
- Seminar (in groups of 3-4) can be chosen from list on course webpage or your own topic.
- Course webpage:
  `www.csa.iisc.ac.in/~deepakd/atc-2024/`
- Teaching assistants for the course: Abhishek Uppar.