Overview of
E0222: Automata and Computability

Deepak D'Souza

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

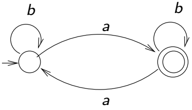
August 5, 2013

What this course is about

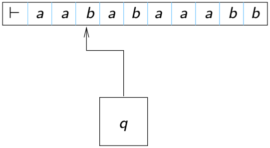
What we study

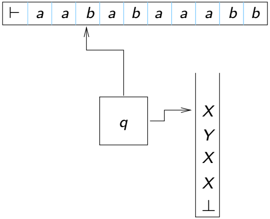
- Models of computation and their expressive power.
- Formal notion of an "algorithm" and "computable" functions.
- Theory of computability.

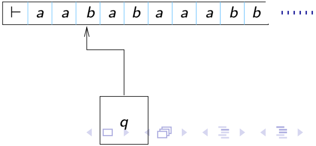
Different State Machines



- Finite-State Automata
- Pushdown Automata
- Turing Machines







Kind of results we study

- Expressive power of the models.
    - What kind of languages do they recognize?
    - What kind of languages can they not recognize?
- Characterisations of the class of languages they recognize:
    - Myhill-Nerode theorem.
    - Büchi's logical characterisation.
    - Parikh's characterisation of Context-Free Languages.
- Existence of languages even Turing machines cannot recognize (undecidable languages).

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. Mathematical Logic (decision procedures for logical problems).
4. Complexity Theory (algorithmic hardness of problems)
5. Formal Verification
   - Is $L(\mathcal{A}) \subseteq L(\mathcal{B})$?
6. Program Analysis

Why a language-theoretic view?

We usually study these machine models in terms of the languages they accept.

- This generalises problems like reachability and satisfaction of linear-time properties, for computer systems (programs, protocols, circuits) which can be modelled using these classes.
- Notion of computability (function f is computable iff its induced language L_f is computable/recursive).

Uses in Verification and Logic

1. Useful problems for Verification
   - Is $L(\mathcal{A})$ empty?
   - Is $L(\mathcal{A}) \subseteq L(\mathcal{B})$?
   - Is a particular configuration of a pushdown automaton reachable?
2. System models are natural extensions of automata
   - Programs with no dynamic memory allocation, no procedures = Finite State systems.
   - No dynamic memory allocation = Pushdown systems.
   - General program = Turing machine.
   - Programs with integer variables = counter machine.
3. Decide satisfiability problems for logics by translating them into automata.

Uses in computability

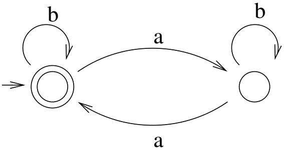
- Notion of a function being computable.
- Formalize the notion of an "algorithm" or a program.
- Connection with language recognition
  - A function f can be represented as a language

$$L_f = \{u\#v \mid f(u) = v\}.$$

  - f is computable iff L_f is decidable.
- Existence of "uncomputable" or "unsolvable" problems.
  - Does a given C program ever terminate?

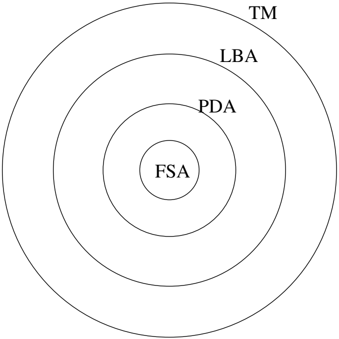
Overview of what we do

Machines and grammars:

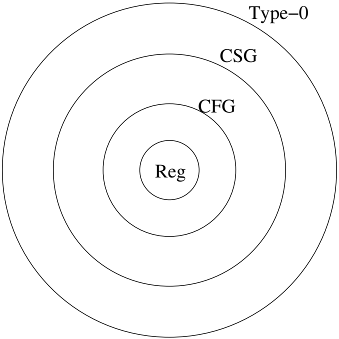


$$S \rightarrow bS \mid aT \mid \epsilon$$
$$T \rightarrow bT \mid aS$$

Overview of what we do



Machines



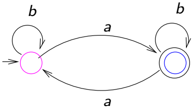
Grammars

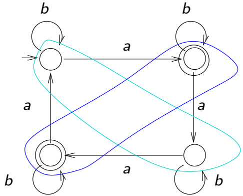
The Chomsky Hierarchy

Some topics which may be new to you

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.

Büchi's logical characterisation of automata

- Describe properties of strings in a logical language
  Eg. "For all positions x in a word which are labelled a, there is a later position labelled b"

$$\forall x (Q_a(x) \Rightarrow \exists y (y > x \ \& \ Q_b(y))).$$

- Büchi's result:

  A language is regular iff it is definable by a sentence in this logic.

First-Order logic of $(\mathbb{N}, <)$.

- Interpreted over $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.
- What you can say:

$$x < y, \quad \exists x \varphi, \ \forall x \varphi, \ \neg, \ \&, \vee.$$

- Examples:
  1. $\forall x \exists y (x < y)$.

First-Order logic of $(\mathbb{N}, <)$.

- Interpreted over $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.
- What you can say:

$$x < y, \quad \exists x \varphi, \; \forall x \varphi, \; \neg, \; \&, \vee.$$

- Examples:
  1. $\forall x \exists y (x < y)$.
  2. $\forall x \exists y (y < x)$.

First-Order logic of $(\mathbb{N}, <)$.

- Interpreted over $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.
- What you can say:

$$x < y, \quad \exists x \varphi, \ \forall x \varphi, \ \neg, \ \&\ , \vee.$$

- Examples:
  1. $\forall x \exists y (x < y)$.
  2. $\forall x \exists y (y < x)$.
  3. $\exists x (\forall y (y \leq x))$.

First-Order logic of $(\mathbb{N}, <)$.

- Interpreted over $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.
- What you can say:

$$x < y, \quad \exists x \varphi, \ \forall x \varphi, \ \neg, \ \& \ , \vee .$$

- Examples:
  1. $\forall x \exists y (x < y)$.
  2. $\forall x \exists y (y < x)$.
  3. $\exists x (\forall y (y \leq x))$.
  4. $\forall x \forall y ((x < y) \implies \exists z (x < z < y))$.

First-Order logic of $(\mathbb{N}, <)$.

- Interpreted over $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.
- What you can say:

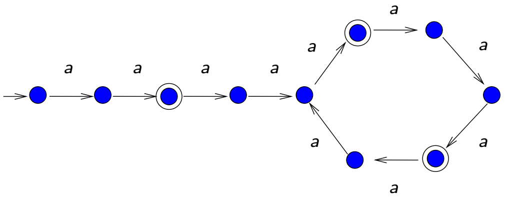
$$x < y, \quad \exists x \varphi, \ \forall x \varphi, \ \neg, \ \& \ , \vee.$$

- Examples:
  1. $\forall x \exists y (x < y)$.
  2. $\forall x \exists y (y < x)$.
  3. $\exists x (\forall y (y \leq x))$.
  4. $\forall x \forall y ((x < y) \implies \exists z (x < z < y))$.
- Question: Is there an algorithm to decide if a given FO$(\mathbb{N}, <)$ sentence is true or not?

Ultimate periodicity of regular languages

Ultimate periodicity

For any regular language L, $len(L) = \{|w| \; : \; w \in L\}$ *is ultimately periodic.*



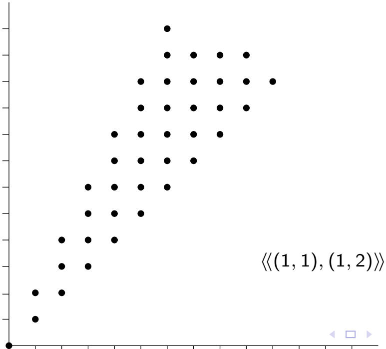
Show properties like “There exist languages L such that neither L nor its complement contain an infinite regular language.”

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).



Reachable configurations of a Pushdown automaton

The set of reachable configurations of a Pushdown automaton is regular.

Useful for program analysis and verification of pushdown systems.

Rice's Theorem

Every non-trivial property of languages accepted by Turing Machines is undecidable.

Can show that checking whether a given TM accepts a regular language is undecidable.

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"

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

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 + 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 pairs) can be chosen from list on course webpage or your own topic.
- Course webpage:
  `www.csa.iisc.ernet.in/~deepakd/atc-2013/atc.html`
- Teaching assistants for the course: Himanshu Arora and Rashmi Mudduluru.
- Those interested in crediting/auditing please send me an email so that I can add you to the course mailing list.