Theoretical Computer Science

June 19, 2022

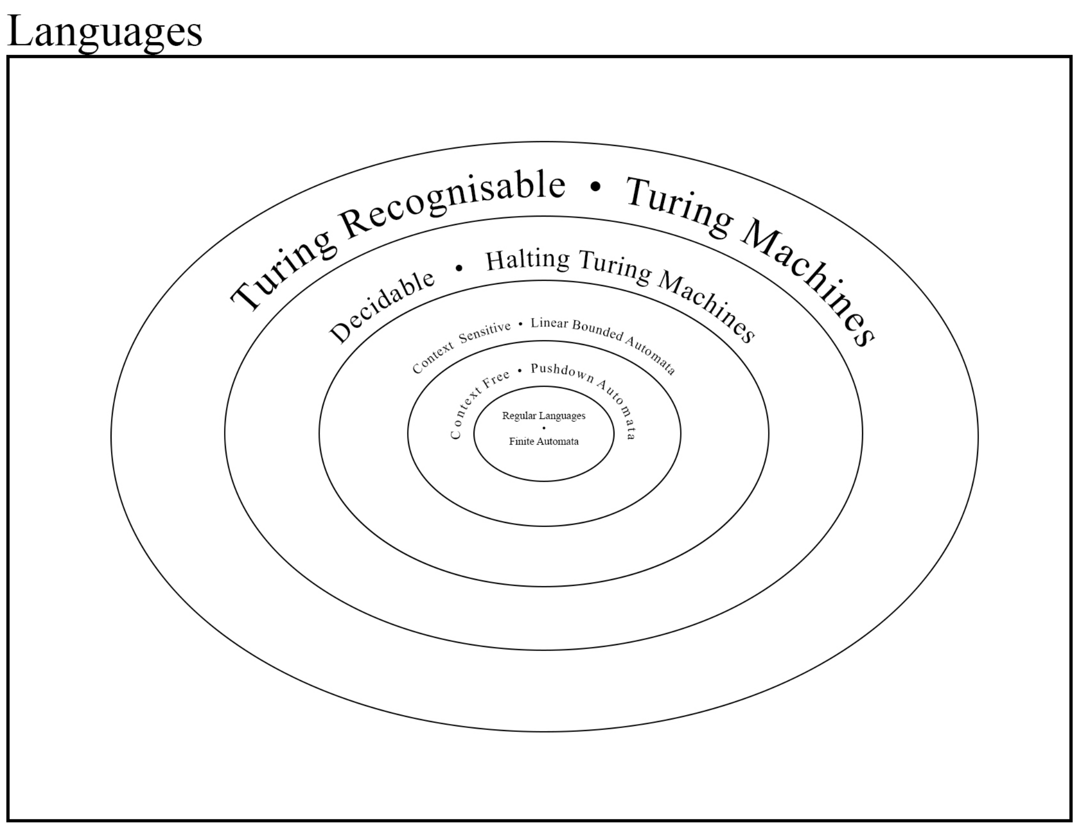
Contents

Chapter 1

Models of Computation

1.1 Preamble

We will examine formal languages, where a language is considered to be a set of strings. Languages will then be classified according to the rules (grammars) that are used to generate strings. Once we have an understanding of the language we can look at a given machines ability to recognise the language (its stings).


Languages
Turing Recognisable • Turing Machines
Decidable • Halting Turing Machines
Context Sensitive • Linear Bounded Automata
Context Free • Pushdown Automata
Regular Languages • Finite Automata

1.2 Propositional Logic

1.2.1 Syntax

Propositional formulas are built from the following set of symbols: Propositional letters $A, .., Z, \phi, ...$; Connectives $\neg, \wedge, \vee, \implies, \iff, \oplus$; Special characters $0, 1, (,)$.

The grammar of propositional logic can be given as context free BNF

$$A, B ::== A|B|0|1|\neg A|A \wedge B|A \vee B|A \implies B|A \iff B|A \oplus B$$

The following are some converntions that allow us to simplify the notation without introducing ambiguity

- Outer most parenthesis are neglected
- An "order of operations" (how tightly each operation binds) is given, from tightest binding to loosest, $\neg, \vee \& \wedge, \oplus, \implies \& \iff$

This allows us to write without introducing ambiguity

$$((P \wedge (\neg Q)) \implies (P \vee (P \iff Q))) \equiv P \wedge \neg Q \implies P \vee (P \iff Q)$$

There are some other common connectives, namely 'nor' $\downarrow$, 'nand' (Sheffer's Stroke) $\uparrow$ and 'if - then - else'.

1.2.2 Semantics

We can assign to each propositional letter a truth value $0, 1$. The semantics of the connectives can then be given by a truth table. This gives an interpretation of any well formed formula as either true or false.

Validity and Unsatisfiability A formula in this language is valid if no truth assignment makes it false. If there exists a truth assignment making it false then it is not valid. A valid formula in propositional logic is a tautology.

Similarly a formula is unsatisfiable if no truth assignment makes it true. If there is some assignment making it true then it is satisfiable. An unsatisfiable formula is a contradiction.

Validity and unsatisfiability properties are preserved under substitution of propositinal letters with formulas. i.e. Taking a valid formula and replacing all propositional letters with formulas will result in a valid formula again.

Proof. If no truth assignment makes the proposition Q false then in particular the possible truth assignments of the formulas that you substitute into Q will not make it false and so the overall statement is still valid. □

The property of unsatisfiability follows because a formula is unstatisfiable $\iff$ its negaition is valid:

Proof.

$$\begin{aligned}\neg Q \text{ valid} &\iff \neg Q \text{ is a tautology}\\ &\iff Q \text{ is a contradiction}\\ &\iff Q \text{ is unsatisfiable}\end{aligned}$$

□

Models, Consequence and Equivlience For a truth assignment θ and a propositional formula Q we say that if $\theta(Q) = 1$ (that is θ makes the propositional statement Q true) then θ is a model of Q. Theta is a function that takes a propositional formala and returns a truth value. Think about it as assigning truth values to all atomic letters and then evaluating the formula.

P is a logical consequence of Q if every model of Q is a model of P. i.e. $\theta(Q) = 1 \implies \theta(P) = 1$. We denote this $Q \vDash P$.

If $Q \vDash P$ and $P \vDash Q$ (they have exactly the same models) then we write $P \equiv Q$.

Just the same as validity, equivilence of two propositional formulas is preserved under substitution of a propositional letter with a more complex formula.

$$\phi \equiv \psi \implies \phi[x \setminus y] \equiv \psi[x \setminus y]$$

More strongly than this is that replacing equivilent formulas with one another preserves logical equivilence, logical consequence, validity, unsatisfiability and even satisfiability. More concretely if ϕ is a subformula of γ and $\phi \equiv \psi$ then $\gamma \equiv \gamma[\phi \setminus \psi]$.

1.3 Predicate Logic

This extends propositional logic, allowing for finitely expressable statements that deal with infinitely many objects. It occomplishes this with variables and quantifiers. Propositional letters are generalised to predicates, functions that map tuples to truth values.

1.3.1 Syntax

1.3.2 Semantics

1.4 Resolution Algorithm

1.4.1 Propositional Logic

1.4.2 Predicate Logic

1.5 Mathematics

1.5.1 Induction

1.5.2 Set Theory

1.6 Regular Languages and Finite Automata

Definition 1

A finite Automata is a five tuple $(Q, \Sigma, \delta, q_0, F)$ representing

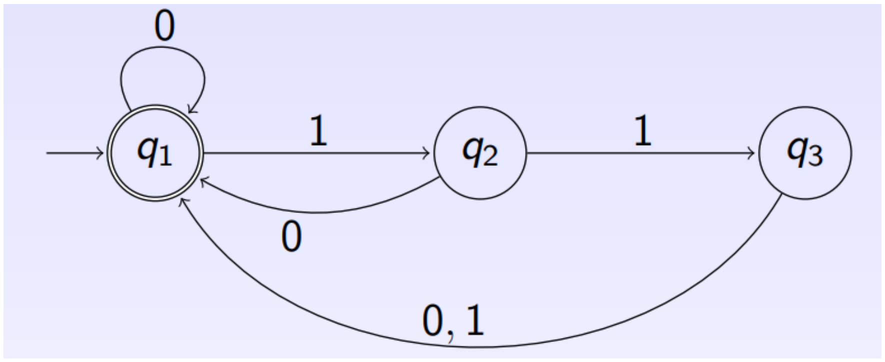
- Q is a finite set of states
- Σ is a finite alphabet (any nonempty finite set)
- $\delta : Q \times \Sigma \to Q$ is a transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ are the accept states (states that the automata can end on)

A string over the alphabet Σ is a finite sequence of symbols from the alphabet. We denote the empty string by ϵ. A language over an alphabet is a set of finite strings over the alphabet (note that the set itself may or may not be infinite). We use Σ^* to denote the set of all finite strings over Σ, this is known as the Kleene star.

Example Automata We can describe a small finite automata $M_1 = (\{q_1, q_2, q_3\}, \{0, 1\}, \delta, q_1, \{q_1\})$ where δ is described by the table

δ	0	1
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_1	q_1

This can be represented as the graph



The language that this automata recognises is

$L(M_1) = \{\epsilon,$ a string ending in 0, a string ending in a sequence of 1's that has length a multiple of 3$\}$

1.6.1 Recognition

We say that an automata $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $w = v_1...v_n \in \Sigma^*$ iff $\exists r_0, ..., r_n \in Q$ (a sequence of states) such that

- $r_0 = q_0$
- $\delta(r_i, v_{i+1}) = r_{i+1}$ for $i = 1, ..., n-1$
- $r_n \in F$

This can be rephrased to say that the automata accepts the string iff the string starts at the automatas starting point, ends at the automatas accept states and only makes a valid transition given by the automatas δ. In other words the string corrosponds to a sequence of valid transitions in the automatas diagram.

The automata recognises a langauge A iff $A \subseteq \{w \in \Sigma^* : M$ accepts w$\}$

1.6.2 Regular languages

A language is regular iff there is some finite automata that recognises it.

For any two languages, A & B, we define the following regular operations

- union
- concatenation $A \circ B = \{xy : x \in A, y \in B\}$
- Kleene Star $A^* = \{x_1...x_k : k \geq 0, x_i \in A\}$
- Intersection
- Compliment
- Set difference i.e. $A \setminus B$
- Reversal

The $k = 0$ case of the Kleene star always corrosponds to ϵ (the empty string).

Lemma 1

Regular languages are closed under regular operations

Proof. The proof is delayed until we have discussed non-deterministic automata □

1.6.3 Nondeterministic Finite Automata

These automata are not actually any different to finite automata (they recognise the same languages) however they are often simpler to reason about. The formalism is

Definition 2

A nondeterministic finite Automata is a five tuple $(Q, \Sigma, \delta, q_0, F)$ representing

- Q is a finite set of states
- Σ is a finite alphabet (any nonempty finite set)
- $\delta : Q \times \Sigma_\epsilon \to Q$ is a transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ are the accept states (states that the automata can end on)

Where $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

Accepance anrecognition are identically defined.

We can think about non-deterministic FA as allowing transitions that dont consume input (ϵ transition), have multiple possible transitions from a state given the same input and have states that one cannot transition out of.

Theorem 1

Every NFA has an equivilent DFA

Proof. Lecture 15 pg 282, all of these proofs are very much unsatisfactory, maybe look up some more detailed ones. We give an algorithm for constructing a DFA from a NFA: □

Now the proof that regular languages are closed under regular operations

Proof. □

1.6.4 Equivilent DFA

For a given regular language we can always find a minimal DFA, that is a DFA that has the smallest possible number of states.

Is the minimal one unique. I imagine not

This suggests a method for testing the equivilence of two DFA, we minimize them both.

Minimizing Algorithm

This algorithm takes a NFA and produces an equivilent, minimal DFA. (note that DFA are a subset of NFA and so this also works for DFA).

1. Reverse the NFA
2. Determinize the result
3. Reverse the result
4. Determinize again

Reversing A NFA To reverse an NFA A with inital state q_0 and accept states $F \neq \emptyset$ we

1. Add a new state q and define it to be the only accept state of the NFA. Then for each state in F add an ϵ transition to q.
2. Reverse all transitions in the resulting NFA. This makes q the initial state and q_0 the only accept state

Note if F is already a singleton set the first step is redundent.

Index