CS312 Notes

August 22, 2024

Contents

I Theory Of Computation Introduction

The 3 componenets of problem solving

1. Unknowns
2. Data
3. Conditions

To solve a problem we need to find a way of determining the unknowns from given data such that conditions of the problem are satisfied.

The traditional areas of the theory of computation (TOC)

- Automata
  - Provide problem solving devices
- Computability
  - Provide framework that can characterize devices by their computing power
- Complexity
  - Provide framework to classify problems acording to time/space complexity of the toold used to solve them

Automata (Automaton)

- Abstration of computing devices
- How much memory can be used?
- What operations can be performed?

Computability

- Study different computing models and identify the most powerful ones
- Range of problems
- Problems can be undecidable or uncomputatble
  - The halting problem

Complexity

- Computing problems range from easy to hard; sorting is easier than scheduling
- Question
  - What makes some problems computationally hard or others easy?

Problem Abstration

Data

- Abstracted as a word in a given alphabet

Conditions

- Abstracted as a set of words called a language

Unknowns

- A boolean variable: true if a word is in the language or false other wise

Abstration of Data

- Σ: alphabet, a finite, nonempty set of symbols
- Σ^*: all words of a finite length built up using Σ
- Rules: (1) the empty word (ϵ) is in Σ^*; (2) if w $\in \Sigma^*$ and a $\in \Sigma$, then aw $\in \Sigma^*$, and (3) nothing else is in Σ^*

Example: If $\Sigma = \{0,1\}$, then $\Sigma^* = \{\epsilon,0,1,00,01,10,11,000,001,010,011,\ldots\}$.

1. Valid C

```
int my_func() { return 1; };

int main() {
    int var = my_func(1,2,3,4,5,6,7);
    for (;;) {}
    // You cannot just simply change the syntax of a for loop
    for(;) {}
}
```

2. Invalid C++

```
int my_func() { return 1; };

int main() {
    int var = my_func(1,2,3,4,5,6,7);
    for (;;) {}
    // You cannot just simply change the syntax of a for loop
    for(;) {}
}
```

II Finite Automata

Formal Language

- Some set of strings over a give alphabet
- How do you specify a language?
- How do you recognize strings in a language?
- How do you translate the language?

Abstraction of Problems

1. Data - word in a given alphabet
   - Σ alphabet, a finite non-empty set of symbols
   - Σ^* all words of finite length built-up using Σ
2. Conditions - Set of words called a language
   - Any subset $L \subseteq \Sigma^*$ is a formal language
3. Unknown - a boolean variable that is true, if word is in language; false, otherwise.
   - Given $w \in \Sigma^*$ and $L \subseteq \Sigma^*$, is $w \in L$?

Formal Definition

- Simplest computational model also referred to as a finite-state machine or finite automaton (FA)
- Representations: graphical, tabular, and mathmatical
- A finite automaton is a 5-tuple (Q,Σ,δ,q_0,F), where Q is a finite set of states, Σ is a finite set of symbols (alphabet), the transition function δ maps Q X Σ to Q, $q_0 \in Q$ is the start (initial) state, and $F \subseteq Q$ is the set of accept (final) states
- Used to design embedded systems, or compilers

Example

If the machine is in a start state, where the initial state is an accept state, that means that our FA can accept an empty string ϵ

DFA

Deterministic Finite Automata

Applications

- Parsers for compilers
- Pattern recognition
- Speech processing and OCR
- Financial planning and market prediction

FA Computation

- Automaton M_1 receives input symbols one-by-one (left to right)
- After reading each symbol, M_1 moves from one state to another along the transition that has that symbol as its label
- When M_1 reads the last symbol of the input, it produces the output: accept if M_1 is in an accept state, or reject if M_1 is not in an accept state

Language Recognition

- If L is the set of all strings that an FA M accepts, we say that L is the language of the machine M and write L(M) = L
- An automaton may accept several strings, but it always recognizes only one language
- If a machine accepts no strings, it still recognizes one language, namely the empty language 0

The machines are recognizing words in the language
Any given automaton only recognizes specifically one language

Formal Definition of Acceptance

- LEt $M = (Q,\Sigma,\delta,q_0,F)$ be an FA and $w = a_1a_2\ldots a_n$ be a string over Σ. We say M accepts w if a sequence of states $r_0r_1\ldots r_n$ exist in Q such that
  - $r_0 = q_0$ (where machine starts)
  - $\delta(r_i,a_{i+1}) = r_{i+1}$, $i=0,1,\ldots,n-1$,(transitions based on δ)
  - $r_n \in F$ (input accepted)

Regular Languages

- We say that FA recognizes the language L if $L = \{w \mid M \text{ accepts } w\}$
- A language is called a **regular** language, if there exists an FA that recognizes it
- Q: how do you design/build an FA

FA Design Approach

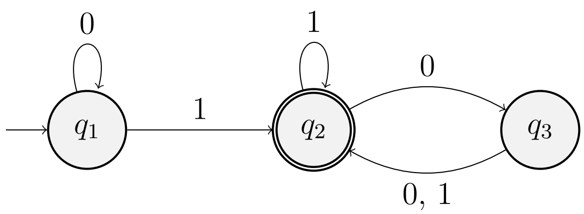
1. Identify finite pieces of information you need, i.e., the states (possibilities)
2. Identify the condition (or alphabet) to change from one state to another
3. Idenitfy the starting and final/accept states
4. Add missing transitions

Example

Let $M_1 = (Q,\Sigma,\delta,q_1,F)$, $Q = \{q_1,q_2,q_3\}$, $\Sigma = \{0,1\}$, and $F = \{q_2\}$. Let's define a transition functoin δ for M_1 and then draw the resulting (graph-based) **state transition diagram** for M_1

DFA, this table is Q X $\Sigma \rightarrow$ Q
q_1 is the start state
q_2 is the accept

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2



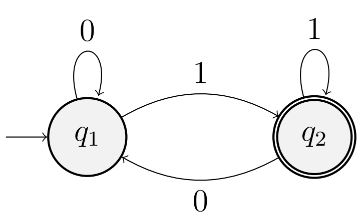
Notes on Example

$L(M_1) = ?$
$L(M_1) = A$
A = {w | w contains at least one 1 AND an event number of 0's following the last 1}

Example 2

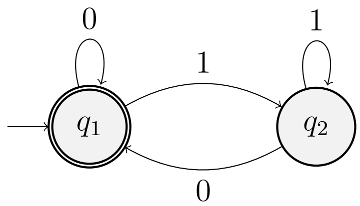
δ Q X $\Sigma \rightarrow$ Q

	0	1
q_1	q_1	q_2
q_2	q_1	q_2



$L(M_2)$ = B = { w | w ends in a 1 }

Expanstion on Above M_3



Language of M_3 = C = { w | w ends in a 0 OR w is empty }

What does this give us?

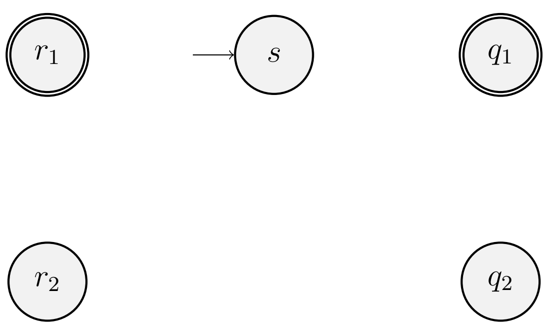
If we flip the accept and initial state, we generate the complement of the machine (flip the meaning)

Last DFA Example

Q={s,q_1,q_2,r_1,r_2}
Σ={a,b}
F = {q_1,r_1}

Δ chart

	a	b
s	q_1	r_1
q_1	q_1	q_2
q_2	q_1	q_2
r_1	r_2	r_1
r_2	r_2	r_1



{ w | starts with 'a' AND ends with 'a' }