FINITA AUTOMATA

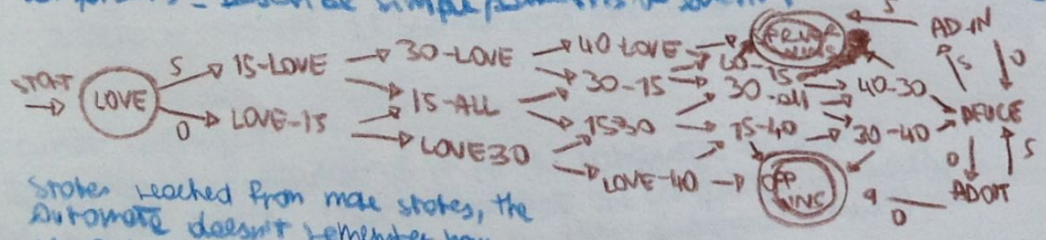
- Mathematical model, like a graph or a robot
- Stores only a finite amount of information

We can do a number of things with finite automata that we can't do with programs, such as what it does or whether there's a shorter program for that

In automata we'll draw whether there are input sequences that bring the automata to an error state or just to check whether systems have flaws

- It is built around a finite collection of states each has a name that represent its history
- State changes in response to inputs (characters or events)
- Rules that tell how the state changes are called transitions

It is useful in things such as design and verification of communication protocols, in text processing applications. They're also an important component of compilers. Describe simple patterns of events



States reached from more states, the automata doesn't remember how it got there (like 15-all)

Acceptance of inputs

- Given a sequence of inputs, start in the start state and follow the transition from each symbol
- We accept a sequence if we end up in the final state after processing an input

The set of strings accepted are A, a language. The language - accepted is denoted by L(A)

...

- Different final states → different languages (i.e. only server winning is final state, the language will be just '5')

- An alphabet is simply a set of symbols
- A string is a sequence of symbols chosen from an alphabet Σ
- Σ^* is the set of strings over that alphabet
- The length of a string is the number of positions in that string
- ϵ represents an empty string

ie $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

we don't make distinction between 0 and '0', but some programming languages do make this distinction '0' for char and "0" for string

- Languages are set of strings and they can be finite or infinite sets, if there is some alphabet that is finite set of symbols from which all strings are composed. A language is a subset of Σ^*

Deterministic Finite automata

- Finite set of states (Q typically)
- An input alphabet (Σ typically)
- A transition function (δ typically)
- A start state (q_0 in Q typically)
- A set of final states ($F \subseteq Q$ typically) /accepting

the transition function takes two arguments: state and symbol.

$\delta(q, a)$ returns the state the automata goes when it is in state q and receives a.

Deterministic → that function has a value for each state and symbol. There the situation where we don't want to continue, in that situation introduce a dead state.

A dead state is a state that is not accepting and has a transition to itself on every input

We can also represent automaton as a table

each row correspond to a state and the column to the input samples. We indicate final states by putting a star next to their names and the start by an arrow.

The entries are the value of δ applied to the state in row and the symbol in the column

"We use lowercase for ~~strings and uppercase for characters~~ letters at the end of the alphabet to represent string. A lowercase letter near the beginning of the alphabet will represent single symbols"

Extended transition function

it takes a state and a string w of any length, and tells you where the automata goes to

$\hat{\delta}$ or δ means extended version.

It is an induction on the length of w

BASIS $\delta(q, \epsilon) = q$

INDUCTION $\delta(q, wa) = \delta(\delta(q, w), a)$

Automata of all kinds define language

- If A is an automaton, L(A) is the language defined by A
- For a DFA A, L(A) is the set of strings labeling paths from start state to final state

(Formally L(A) is the set of strings w such that $\delta(q_0, w)$ is in F)

A set former, it describes sets, in particular languages here.

is $\{w \mid$ SUCH THAT w is in $\{0,1\}^*$ and w does not have two consecutive zeros$\}$

THE SET CONSISTS OF SOME STRINGS W

WHAT MUST BE TRUE TO BE A MEMBER OF THE SET

...

...

Often we want to prove that two [illegible] descriptions of sets are the same set

To prove two sets S and T are equal we generally need to prove that each is contained in the other

given S the language of this DFA and T the set of strings with [illegible] no consecutive 1's

proof by induction on the length of w

HINT: By making a more detailed statement here, will make the inductive proof work

Inductive hypothesis
1 If w gets you to state A, it does not have one.
2 If w ends in B is still good, but ends in one.

Basis $|w|=0$

1 Holds since ϵ has no ones
2 Holds since $\delta(A,\epsilon)$ is not B — vacuously (if (false) then false true? true)
($\delta(B,w)$ is false and ϵ doesn't end in one, so since false implies false, that is true)

assume that $|w|$ is at least one. Since w is not empty we can write $w=xa$ (x string and a last symbol)

Inductive step
1 since everything going to A is 0, w cannot end in one
2 By IH, x has no 11's
3 Thus w has not 11 and does not end in 1

...

Regular languages

A language is regular if it is the language accepted by some DFA. That means that the language is exactly the set of strings accepted by this automata (we'll see other ways to identify them [illegible])

Some languages are not regular, intuitively, regular languages cannot count to arbitrarily high integers, thus cannot do things like check whether they have seen the same number of zeros as ones on their input, or check that parenthesis are balanced. For we need more powerful mechanisms such as context-free grammars

...

... eg of non-regular languages

$L_1 = \{0^n 1^n \mid n \geq 1\}$ $\quad L_2 = \{w \mid w \text{ in } \{(,)\}^* \text{ and } w \text{ is balanced}\}$

$(0^4 = 0000)$

$L_1 = 000111$

eg of regular languages

- Format for floating point numbers

$-L_3 = \{w \mid w \text{ in } \{0,1\}^* \text{ and } w \text{ viewed as binary is divisible by } 23\}$

The DFA will have
- 23 states (0 to 22)
- correspond to remainders
- start state is 0 because we interpret the empty string as 0, it is also the final state

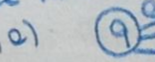
Transitions
w represents integer i
$\delta(0,w) = i \% 23$

The transition from each state i on input zero, is to the state that is the remainder of $2i/23$

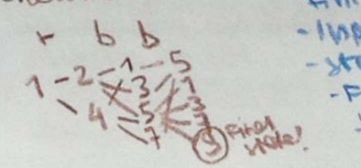
$i = 23a + b$

If a language is regular, then its reverse is also regular.

Nondeterministic Finite Automata

- It can be in several states at once
- $\delta(q,a)$  it guesses, goes to all those states or one
- Has a start state and can have any # of final states
- Input that leads to final state is accepted the NFA is allowed to guess which way to go and it always guesses right no matter how wrong guesses it also makes

chessboard



...

$\delta(q,a)$ is a set of states
Extended (to string)
basis $\delta(q,\epsilon) = \{q\}$
induction $\delta(q,wa) = \delta(q,w) \cup \delta(q,a)$

- The language of a NFA is the set of strings that it accepts

The language accepted by an NFA (is the) a DFA that accepts the same language
(Proof is the subset construction

however the # of states of the DFA can be exponential the # of states of the NFA. But that is not always the case

proof next page

NFA to construct equivalent DFA
- with states 2^Q (power set of Q, the set of subsets of Q)
- input Σ
- start $\{q_0\}$ (set containing start of NFA state)
- final = F
- transition function $\delta_D(\{q_1 \ldots q_k\}, a)$ single state is the union over all $i=1..k$ of $\delta(q_i, a)$

NFA components (much like DFA)
· Finite # states, Q
- Input alphabet, Σ
- start state, q_0
- Final state, $F \subseteq Q$
the big difference is δ ...

eg checkerboard

For strings consisting only of b's is easy to describe the language, only even-length, non empty strings are accepted

subset construction

Given an NFA with states Q, inputs Σ, transition δ_N, start q_0, and final states F. Construct a DFA with:

- states 2^Q (power set, set of all subsets of Q) — if the set has N elements, the power set has 2^N elements
- inputs Σ (same as NFA)
- start $\{q_0\}$ set containing the q_0 of NFA.
- Final states = all those with a member of F.

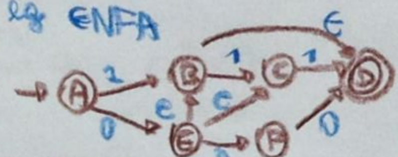
" DFA states have names that look like a set of states, but they're single objects."

$\delta_D(\{q_1 \dots q_k\}, a)$ is the union over all $i = 1 \dots k$ of $\delta_N(q_i, a)$

The NFA can make transitions on epsilon. That is, without taking any input.

This new model (ε-NFAs) can only accept regular languages, like NFAs.

eg ENFA



ε is not an input symbol. Not member of Σ

From E, if the input is 1, we can spontaneously go to B and then follow the 1 to C, or do the same from C.

to convert from ε-NFA to NFA we introduce the notion of closure.

the closure of a state q (CL(q)) is the set of states we can get from q and following only ε transitions

eg. CL(A) = {A}

CL(E) = {B, C, D, E}

the closure of a set of states is just the union of the closure of each of those states

$\hat{\delta}(q, w)$

BASIS $\hat{\delta}(q, \epsilon) = CL(q)$

IND $\hat{\delta}(q, xa)$

- $\hat{\delta}(q, x) = S$
- union of $CL(\delta(p, a))$ for all p in S

$\hat{\delta}(A, \epsilon) = CL(A) = \{A\}$

$\hat{\delta}(A, 0) = CL(\{E\}) = \{B, C, D, E\}$

$\hat{\delta}(A, 01) = CL(\{B, D\}) = \{C, D\}$

language of an ENFA is the set of strings w such that $\delta(q_0, w)$ contains a final state

- A language accepted by an NFA is also accepted by an ENFA.
- Every NFA is an ENFA with no transitions on ε.
- We get rid of ε transitions by combining with the next transition on a real input

Only a DFA can be implemented!

(there no such a thing as a non deterministic computer)

REGULAR EXPRESSIONS

Algebraic notation that describes a regular language. To describe patterns in text

They can exactly describe a regular language.

Given a regular exp. E, L(E) is the language it defines.

It has three specialized operators

Union → Union is the same in text since languages are sets

eg $\{01, 111, 10\} \cup \{00, 01\} = \{01, 111, 10, 00\}$

concatenation → two languages, L and M, concatenated, are denoted as LM.

$\{01, 111, 10\}\{00, 01\}$ = [illegible]

Kleene star → given language L, L^* is the set of strings formed by concatenating zero or more strings from L [illegible]

$L^* = \{\epsilon\} \cup L \cup LL \cup LLL \dots$

$\{0, 10\}^* = \{0, \epsilon, 10, 00, 010, 100, 1010 \dots\}$

Regular expression definition

BASIS 1 If a is a symbol, then a is a RE, and $L(a) = \{a\}$

BASIS 2 ε is a RE and $L(\epsilon) = \{\epsilon\}$

BASIS 3 ∅ is a RE, and $L(\emptyset) = \emptyset$

INDUCTION 1 If E_1 and E_2 are REs, then $E_1 + E_2$ is a RE, and $L(E_1 + E_2) = L(E_1) \cup L(E_2)$

INDUCTION 2 If E_1 and E_2 are REs, then $E_1 E_2$ is a RE, and $L(E_1 E_2) = L(E_1)L(E_2)$

INDUCTION 3 If E is a RE, then E^* is a RE and $L(E^*) = (L(E))^*$

= parentheses can be used to force precedence

- highest precedence is $*$, then concat, then +

Example RE's

$L(01) = \{01\}$ — concatenation of language consisting of the symbol '0' and of the language consisting of the symbol '1'

$L(01, 0) = \{01, 0\}$ — union of the language containing only 01, and the language containing only 0.

Any string of symbols is a RE containing only that one string

$L(0(1+0)) = \{01, 00\}$

$L(0^*) = \{\epsilon, 0, 00, 000 \dots\}$

$L((0+10)^*(\epsilon+1))$ = all strings of 0 and 1s without two consecutive 1s ...

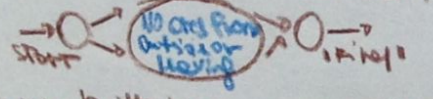
Equivalence of RE's and FA

We already have three equal representations of the regular languages: DFAs, NFAs and GNFAs

- We'll pick the most powerful ε-NFA to show that for every RE there's an automaton
- And the DFA to show that for every automaton there's a RE defining its language

Convert RE to an ε-NFA

The automaton will have this form



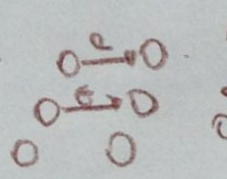
As we build larger automata from smaller ones, we allow only arcs coming to the start state.

And we never have an arc from middle to a state different from final.

The induction is on the number of operators in the RE

BASES CASES

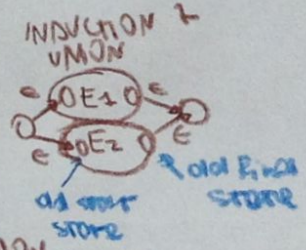
symbol a:



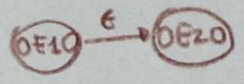
ε

∅

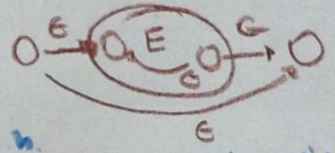
INDUCTION 1 UNION



INDUCTION 2 CONCATENATION



INDUCTION 3 CLOSURE



DFA to RE

- States of DFA named 1..n.
- Induction on the max state k we're allowed to travel in doing the path

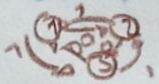
k paths. paths that can go from any state to any state, only in the middle and numbered k or less.

There is a RE whose language is the set of labels of all k paths who take i to j

0 paths from 2 to 3
RE for labels = 0

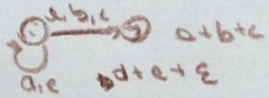
1-paths from 2 to 3
RE for labels = 0+11

2-paths from 2 to 3
RE for labels
$(10)^*0 + 1(01)^*1$



BASIS

k=0 only a single arc, or if i=j, no arc, but add an ε.



Let R_{ij}^k be the RE for the set of labels of k-paths from i to j

INDUCTION k=0. R_{ij}^0 = sum of labels of arcs going from i to j

a k-path from i to j either
- never goes through k or
- goes through k one or more times

$R_{ij}^k = R_{ij}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}$

CONCLUSION the RE with same language as DFA is the sum of R_{ij}^n where
- n is the # of states
- i is the start state
- j is one of the final states

Algebraic laws for REs

- union is commutative and associative
  $x+y=y+x$ $(x+y)+z=x+(y+z)$
- concatenation is associative, but distributes over union. It is NOT commutative
  $x(y+z)=xy+xz$

$R+\emptyset=R$ $\quad \varepsilon R=R$ $\quad \emptyset R=\emptyset$

UNIX RE's

Unix uses REs in many places, including grep command (global RE and print)

Most Unix commands use an extended RE notation, that still defines only regular languages.

Ken Thompson before Unix had worked on a system for processing REs by converting them into a simulated NFA.

UNIX NOTATION

one or more characters

- $[a_1a_2\ldots a_n]$ is shorthand for $a_1+a_2+\cdots+a_n$
- you can describe sequences that are consecutive in the ASCII order by using first-dash-last in brackets.

  eg [a-z] = any lowercase
  [a-zA-Z] = any letter (they're not consecutive in ASCII)

"you can use \ to use " [or - "

- Dot = any character
- | is used for union, instead of +
- But + is used like star and means one or more

  [a-z]+ = one or more lowercase characters

  $E+ = EE^*$ unix notation for that

- ? means zero or one of

  $E? = E+\varepsilon$

  eg [ab] = "an optional a or b"