Lecture 2

Quantifiers & Games $\forall, \exists$

$\forall n \exists m \quad n < m$ ✓

$\exists m \forall n \quad n < m$ ✗

Game: 2 players

Elise, Abelard

when you see a formula each quant. represents a move.

Elise: a value for the variable.

$\exists$ → Elise's move

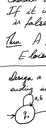
$\forall$ → Abelard's move

At the end verify whether the pairs with the variables belong to the chosen values is true or not.

If it is true Elise wins, if it is false Abelard wins.

Then A formula is valid iff Elise has a winning strategy.

Image a machine that accepts strings starting in "aa" $\Sigma = \{a, b\}$



Non-deterministic Finite Automata

There can be multiple arrows on one input state. This machine makes choices.

There are multiple computation paths corresponding to different choices.

A word is accepted if $\exists$ a path leading to an accept state.

Then The language accepted by any NFA is a regular language i.e. could also have been accepted by a DFA.

Remark There is even an algorithm to convert an NFA into an equivalent DFA.

Another extension: NFA+ε

The machine can make jumps without reading input.

NOT BOTH EMPTY

DFA ≡ NFA ≡ NFA+ε

≡ Regular languages

Ex decimal numbers

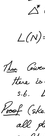
± in front, optional

sequence of digits (perhaps empty)

sequence of digits (perhaps empty)

+ 72.39

+ 13.X, $\pm 3^{\times}.^{\times}4, \pm .5^{\times}5$



NFA+ε

Formal Def of an NFA: Σ

An NFA is a 5-tuple:

(1) Q: a finite set of states

(2) $q_0 \subseteq Q$: a (finite) set of start states

(3) $F \subseteq Q$: accept states

(4) Δ: $Q \times \Sigma \to 2^Q$

NFA+ε

(4') Δ: $Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$

NFA without ε.

Δ^* $Q \times \Sigma^* \to 2^Q$

$\Delta^*(p, \varepsilon) = \{p\}$

$\Delta^*(p, ax) = \bigcup_{q \in \Delta(p,a)} \Delta(q, x)$

$L(N) = \{w \in \Sigma^* \mid \exists q \in q_0, \Delta(q, w) \cap F \neq \emptyset\}$

Then Given an NFA $N = (Q, q_0, F, \Delta)$

there is a DFA $M = (Q, q_0, F', \delta)$

s.t. $L(N) = L(M)$.

Proof (sketch): Idea keep track of all places in Q where the m/c could be.

$S = 2^Q$

$s_0 = q_0$

$F' = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$

$\delta(S, a) = \bigcup_{q \in S} \Delta(q, a)$

It is easy to see that this works as intended. ∎ ☺

Remark It is possible to have an exponential blow-up in the number of states.

For NFA+ε we define ε-closure of a set to be the places one can get to with an ε-move.

L: regular language

$\frac{1}{2}L = \{w \in \Sigma^* \mid \exists x \in \Sigma^*, |x| = |w|, wx \in L\}$

Assume (S, s_0, F, δ) recognizes L

we will construct an NFA for $\frac{1}{2}L$

State space $Q = S \times S \times \Sigma^*$ each = (current DFA state, where we are in the DFA, what remaining input symbols)

$Q_0 = \{(s_0, s, (a))\}$

$F' = \{(s, s, x) \mid x \in F\}$

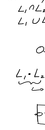
$\Delta((s, s, x), a) = \{(s, s, x')\} \cup \{(s, s, x') \mid x'a = x\}$

x' is the set of states DFA may reach from starting every symbol x'

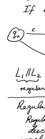
Closure properties of regular languages

$L_1 \cap L_2$ is regular → easy.

$L_1 \cup L_2$ is regular.



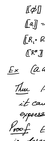
$L_1 \cdot L_2$ is regular



$L^* = \{w \cdots w_k \mid w_i \in L\} \cup \{\varepsilon\}$

Kleene star operator

If L is regular L^* is also regular.



Regular languages are an algebra.

Regular expressions: Notation for describing regular languages via machine-friendly syntax.

if a is a reg exp If a ∈ Σ

ε is a reg exp then a is a reg exp

If R_1, R_2 are regular expressions

$R_1 R_2$, $R_1 + R_2$ are regular expressions

If R is a regular expression R^* is a reg exp

[If R is a regular exp R is a reg exp]

A reg exp denotes a language

$[\![R]\!]$ = the set of strings (words) in the language denoted by R.

$[\![\varepsilon]\!] = \varepsilon$ $[\![a]\!] = \{a\}$

$[\![R_1 R_2]\!] = [\![R_1]\!] \cdot [\![R_2]\!]$

$[\![R_1 + R_2]\!] = [\![R_1]\!] \cup [\![R_2]\!]$

$[\![R^*]\!] = ([\![R]\!])^* / [\![R]\!]$ $\xrightarrow{\text{def}}$ $\bigcup_{i=0}^{\infty} [\![R]\!]^i$

Ex $(a, a+b)^* = \{aa, a, b, aab, bb, aab, \ldots\}$

Then A language is regular iff it can be described by a regular expression. [Kleene Thm]

Proof Easy to see that if a language is described by a reg exp it is regular. Harder to show that every regular language can be described by a reg exp.

Kleene's direction (hard, if time)

Equations between regular expressions

ex $(a+b)^* = (a+b)^* a^*$

We have a set of equational axioms for reasoning about equality of regular expressions.

Reg → set of regular expressions (alphabet Σ)

(Reg, $\oplus$, $\cdot$) → commutative monoid

(Reg, $\cdot$, ε) → monoid

• distributes over + on both sides

$\oplus$ annihilates with $\cdot$

$\oplus \cdot x = x \cdot \oplus = \oplus$

$\varepsilon + x^* x^* = x^*$

$\varepsilon + x^* x = x^*$

We can define ≤ by saying

$x \leq y$ iff $[\![x]\!] \subseteq [\![y]\!]$

$\begin{cases} \oplus + x \leq y \Rightarrow x^* \oplus \leq y \\ \oplus + x \leq y \Rightarrow \oplus^* x \leq y \end{cases}$

MAJOR THM: These rules are complete i.e. any valid equation can be derived from these rules.

REMARK: In order to do anything algorithmic, use DFA.

$(a+b)^* = a^*(ba^*)^*$ → example of valid equation

$(a+b)^* a^* = (a+b)^* a^*$

$(a+b)^* = a^*(ba^*)^*$ for any regular exp. a, b.

From DFA to reg exp: Idea

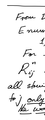
Enumerate the states of a DFA

$1, 2, \ldots, k$

For every pair of states we define $R_{ij}^{(k)}$: a regular expression describing all strings that take the DFA from i to j only traversing states $1, \ldots, k$ along the way.

$R_{ij}^{(k)}$: only direct jumps from i to j

We compute $R_{ij}^{(k)}$ from $R_{ij}^{(k-1)}$ as follows

When we get $R_{ij}^{(k)}$ we have all strings that go from i to j.

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