IX

AUTOMATA & FORMAL LANGUAGES
NINTH LECTORE
2 NOVEMBER 2024

We are in the middle of the proof of Proposition 2.29

There is an algorithm that determines whether two states of an automaton are equivalent.

THE TABLE FILLING ALGORITHM

FROM LECTORE VIII:

Back to the EQUIVALENCE PROBLEM

What does all of this have to do with it?

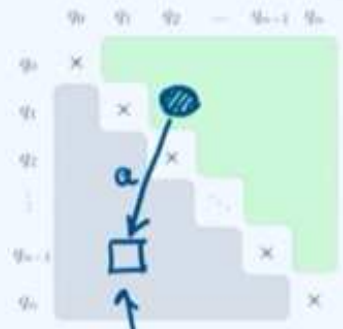
Proposition 2.28. There is an algorithm that determines which states of an automaton are inaccessible.

Proof. Let $D = (\Sigma, Q, \delta, q_0, F)$ and $n := |Q|$. By Corollary 2.14, a state q is inaccessible if and only if there is no word w of length $\leq n$ such that $\hat{\delta}(q_0, w) = q$. Since there are finitely many such words, we can just check $\hat{\delta}(q_0, w)$ for all such words to determine which states are accessible; the remaining states must be inaccessible. Q.E.D.

Proposition 2.29. There is an algorithm that determines whether two states of an automaton are equivalent.

Proof. We determine whether the states are indistinguishable; from this we can easily determine equivalence. This algorithm is known as the *table filling algorithm*. We write $Q \times Q$ as a table; note that due to the fact that indistinguishability is an equivalence relation, we only need to fill half of the table, so we can ignore the lower left triangle.

IDEA:
We mark (q, q') with w if w distinguishes q and q'.



EXAMPLE
$q_2 \xrightarrow{a} q_{n-1}$
$q_1 \xrightarrow{a} q_2$

If (q_{n-1}, q_2) is distinguished by w, then (q_2, q_1) is distinguished by aw!

LECTURE VIII :

Description of the TABLE FILLING ALGORITHM

Remark on P 2.28 This is just the (proof of the) pumping lemma
The shortest word s.t. $\delta(q_0, w) = q$ must have length $< |Q|$. So just check all words of these lengths.

Proof of Prop 2.29 Algorithm: **TABLE FILLING ALGORITHM**

STEP 1 Check all (q,q'). If $q \in F \& q' \notin F$, then write ε in (q,q')
$q \notin F \& q' \in F$

STEP $n \geq 1$ Have partially filled table. Check all unmarked (q,q') and for each $a \in \Sigma$ consider $(\delta(q,a), \delta(q',a))$
If unmarked, do nothing.
If marked by w, write aw in (q,q').
After that, if nothing was marked in STEP n, TERMINATE.
If sth was marked, go to STEP $n+1$.

This terminates, since there are only $|Q|^2$ pairs, so it'll stop before step $|Q|^2 + 1$.

We obtain a partially filled table.

if (q,q') is marked, q, q' are distinguished

Still need to show:

if (q,q') is unmarked, q, q' are indistinguishable

→ SATURDAY.

So, it remains to show:
if (q,q') is unmarked at the end, q & q' are indistinguishable.

Proof of remaining claim

Call (q,q') a bad pair if (q,q') is unmarked but there is a distinguishing q from q'.

Suppose there is one, pick one s.t. the least distinguishing word w is of minimal length n.

Note $n>0$, o/w (q,q') would have been marked in STEP 1.

So $n=k+1$, so $w=av$ for some $a\in\Sigma$ & $v\in W$.

Consider $(\delta(q,a),\delta(q',a))$: this is distinguished by v.

But if it's marked, then so is (q,q')

So $(\delta(q,a),\delta(q',a))$ is a bad pair, but $|v|<|w|$;

Contradiction to min. of w. q.e.d.

Corollary (Thm 2.30) The equivalence problem for det. automata [and thus, for regular grammars] is solvable.

Proof D, D' any two automata.
Use ? 2.28 & 2.29 to construct the irreducible minimal automata I, I' s.t. $\mathcal{L}(D) = \mathcal{L}(I)$ and $\mathcal{L}(D') = \mathcal{L}(I')$.
Now check whether I & I' are isom.
If they have different numbers of states $\longrightarrow$ "No"
If they have the same number, say n, then there are n^n functions between their sets of states. Check all of them whether they're isos. If one is $\longrightarrow$ "Yes"
If not $\longrightarrow$ "no".
q.e.d

Coda to CHAPTER 2 : Regular expressions

Given alphabet Σ, consider

$$\overline{\Sigma} := \Sigma \cup \{\emptyset, \varepsilon, (,), +, ^+, ^*\}$$

Define REGULAR EXPRESSIONS

(1) The symbol $\varnothing$ is a regular expression;

(2) the symbol ε is a regular expression;

(3) every $a \in \Sigma$ is a regular expression,

(4) if R and S are regular expressions, then $(R + S)$ is a regular expression;

(5) if R and S are regular expressions, then (RS) is a regular expression;

(6) if R is a regular expression, then R^+ is a regular expression;

(7) if R is a regular expression, then R^* is a regular expression;

(8) nothing else is a regular expression.

Kleene Star and Plus :

$$L^+ := \{ w ; \exists n>0 \; \exists (w_1, \dots, w_n) \in L^n \; w = w_1 w_2 \dots w_n \}$$
$$L^* := L^+ \cup \{\varepsilon\}.$$

ASSIGN LANGUAGES TO REGULAR EXPRESSIONS :

Examples

$a \rightsquigarrow \{a\}$

$a+b \rightsquigarrow \{a,b\}$

$a^* \rightsquigarrow \{a\}^* = \{a^n ; n \in \mathbb{N}\}$

$a^*+b^* \rightsquigarrow \{a^n ; n \in \mathbb{N}\} \cup \{b^n ; n \in \mathbb{N}\}$

$(a^*+b^*)c \rightsquigarrow \{a^n c ; n \in \mathbb{N}\} \cup \{b^n c ; n \in \mathbb{N}\}$

(1) If $E = \varnothing$, then $\mathcal{L}(E) = \varnothing$;

(2) if $E = \varepsilon$, then $\mathcal{L}(E) = \{\varepsilon\}$;

(3) if $E = a$ for $a \in \Sigma$, then $\mathcal{L}(E) = \{a\}$;

(4) if R and S are regular expressions, then $\mathcal{L}((R + S)) = \mathcal{L}(R) \cup \mathcal{L}(S)$;

(5) if R and S are regular expressions, then $\mathcal{L}((RS)) = \mathcal{L}(R)\mathcal{L}(S)$;

(6) if R is a regular expression, then $\mathcal{L}(R^*) = \mathcal{L}(R)^*$;

(7) if R is a regular expression, then $\mathcal{L}(R^+) = \mathcal{L}(R)^+$.[10]

Theorem TFAE:

(i) L is essentially regular

(ii) there is a regular expression R s.t. $L = \mathcal{L}(R)$

No proof. Note (ii) $\Rightarrow$ (i) is in the typed notes and will be discussed on ES#2.

The direction (i) $\Rightarrow$ (ii) is not in the notes, but instructive examples on ES#2.

Proposition 2.19

CHAPTER 3

CONTEXT-FREE LANGUAGES

Context-free rules

$A \longrightarrow \alpha$ any $\alpha \in \Omega^+$ $\quad A \in V$

Natural language in general is not context-free:

$S \longrightarrow PV$
$P \longrightarrow$ he $\quad P \longrightarrow$ she $\quad P \longrightarrow$ they
$V \longrightarrow$ go $\quad V \longrightarrow$ goes

$S \longrightarrow PV \longrightarrow$ they $V \longrightarrow$ ~~they goes~~ NOT GRAMMATICAL

Natural language is CONTEXT-SENSITIVE.

INSTEAD

he $V \longrightarrow$ he goes
they $V \longrightarrow$ they go
etc.

} These rules fit natural language grammar, but are <u>not</u> context-free.

Examples of c-f grammars & derivations.

$S \longrightarrow ASB$
$S \longrightarrow AB$
$A \longrightarrow a$
$B \longrightarrow b$

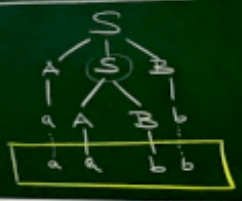
This was our example of a c-f, non-regular language.

$S \longrightarrow ASB \longrightarrow AABB \longrightarrow aABB \longrightarrow aaBB$
$\longrightarrow aabB \longrightarrow aabb$

Note: Very non-unique $\longrightarrow$ the order of the individual derivation steps is almost irrelevant.

This is a consequence of context-freeness

This gives rise to a tree structure



These trees are called PARSE TREES.

These are labelled trees, i.e., a tree T with a labelling function $\ell : T \longrightarrow \Omega \supseteq \Sigma \cup V$.

Def A parse tree starting from A is a labelled tree $T = (T, \ell)$ s.t.

(i) the root of T gets label A;

(ii) $\ell(t) \in \Sigma \iff t$ is a terminal node in T.

If T is a parse tree, we write $\sigma_T \in W$ for the left-to-right reading of the labels of its terminal nodes.

Prop 3.2 If G is context-free! TFAE

(i) $w \in \mathcal{L}(G)$

(ii) there is a parse tree T starting from S, using only G-rules s.t. $w = \sigma_T$.

What's the relationship between parse trees and derivations?

① If $A \xrightarrow{G} w$ is a G-derivation, we can rewrite it uniquely into a parse tree T starting from A s.t. $\sigma_T = w$.

② If T is a parse tree starting from A using only rules from G, then there are (not necessarily unique) derivations $A \xrightarrow{G} \sigma_T$.