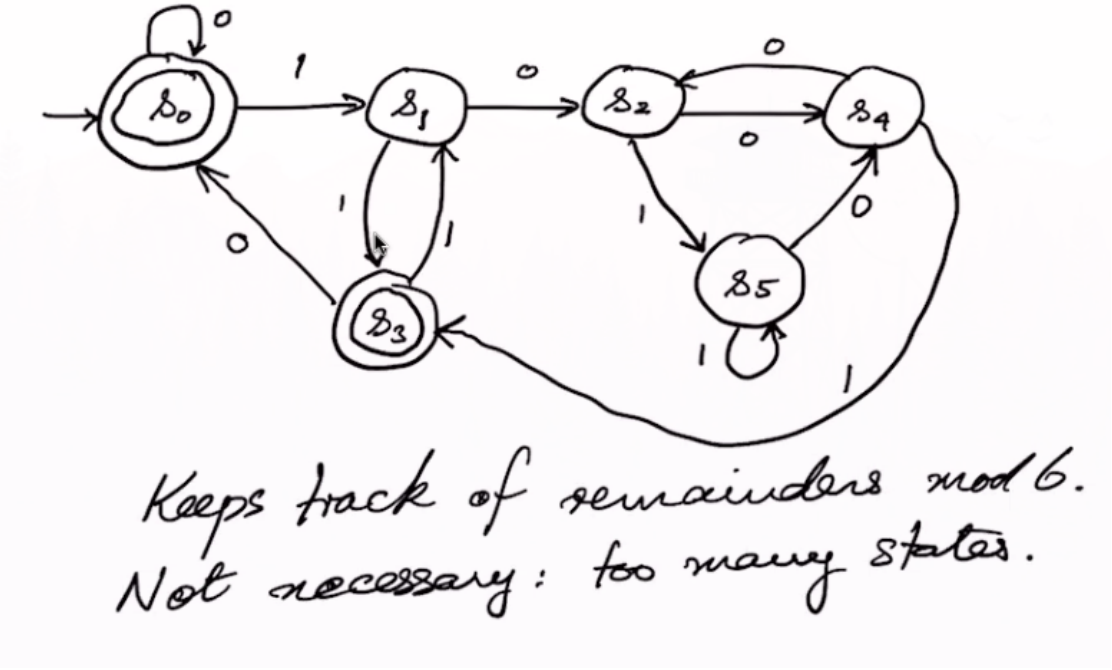
§1 05-06

§1.1 Minimization of DFA

Example of machine that checks divisibility by 3. But we already saw a three state machine that does the same thing.



$$\delta(s_0, 0) = s_0 = \delta(s_3, 0)\delta(s_1, 1) = s_1 = \delta(s_3, 1)$$

These states are doing the same thing so there's no reason to keep them separate.

Note 1.1. Once you get to a state, it doesn't matter how you got there. What happens in the future only depends on what you read in the future.

This is the fundamental meaning of the word state. It encodes the information to predict the future.

Definition 1.2. Given a DFA $M = (S, s_0, \delta, F)$, Σ, p, q are equivalent, $p \approx q$ if

$$\forall x \in \Sigma^*,\ \delta^*(p, x) \in F \Leftrightarrow \delta^*(q, x) \in F$$

So the languages defined by their starting states are the same.

Remark 1.3. $p \not\approx q$ means $\exists x \in \Sigma^*$ such that

$$\begin{gathered}(\delta^*(p, x) \in F \ \wedge \ \delta^*(q, x) \notin F) \\ \vee \\ (\delta^*(p, x) \notin F \ \wedge \ \delta^*(q, x) \in F)\end{gathered}$$

Observe that $\approx$ is an equivalence relation.

Lemma 1.4

$p \approx q \Rightarrow \forall a \in \Sigma\ \delta(p,a) \approx \delta(q,a)$. But not necessarily $\delta(p,a) = \delta(q,a)$.

$$[p] \coloneqq \{q \mid p \approx q\},\ p \approx q \Leftrightarrow [p] = [q]$$

This means that $[p] = [q] \Rightarrow [\delta(p,a)] = [\delta(q,a)]$.

We define a new machine $M' = (S', s_0', \delta', F')$.

$$S' = S/\approx,\ s_0' = [s_0],\ F'\{[s]\ \}$$

This is a simpler draft

Lemma 1.5

$$p \in F\ \wedge p \approx q \Rightarrow q \in F$$

This is trivial when considering $\epsilon \in \Sigma^*$

Lemma 1.6

$\forall w \in \Sigma^*,\ \delta'^*([p], w) = [\delta^*(p, w)]$

Proof. By induction on the length w. □

Theorem 1.7

$L(M) = L(M')$

Proof.

$$x \in L(M') \Leftrightarrow \delta'^*([s_0], x) \in F'$$
$$\Leftrightarrow [\delta^*(s_0, x)] \in F' \Leftrightarrow \delta^*(s_0, x) \in F$$
$$\Leftrightarrow x \in L(M)$$

□

Quotient construction. Might be that every equivalence class contains element, but maybe each one contains several states. But definitely didn't get a bigger machine. This process is called collapsing a machine.

§1.2 Splitting Algorithm

$p \bowtie q$ if $\exists w \in \Sigma^*$ s.t. $\delta^*(p,w) \in F$ & $\delta^*(q,w) \notin F$ OR vice versa.

Fact 1.8. If $\exists a \in \Sigma$ such that $\delta(p,a) bowtie \delta(q,a)$, then $p bowtie q$. The contrapositive of the above.

Matrices of booleans. You can multiply matrices of booleans by using "and" and "or". The collection of $n \times n$ matrices over a ring is always a ring.

Algorithm

1. For every (p,q) such that $p \in F \wedge q \notin F$, put 0 in the (p,q) cell of the matrix.
2. Repeat until no more changes. For each pair (p,q) that is not already marked with a 0, check if $\exists a \in \Sigma$ such that $(\delta(p,a), \delta(q,a))$ is marked with 0. If so mark (p,q) 0.

This algorithm correctly computes the equivalence classes. The time complexity is $O(n^4)$ in the naive case. $O(n^2 k)$ using data structures. $O(n \log n)$ Hopcraft method.

Theorem 1.9

If at the end two states are not marked 0 then they are equivalent.

Theorem 1.10

There is a unique minimal automaton to recognize a regular language.

This is how you prove the equivalence of regular expressions.

Definition 1.11 (Right invariant)**.** An equivalence relation R on Σ^* is said to be right-invariant if

$$\forall x, y \in \Sigma^* \text{ such that } xRy \Rightarrow \forall z \in \Sigma^*,\ azRyz$$

Example 1.12

DFA $M = (Q, \Sigma, q_0, \delta, F)$,

$$xR_m y \Leftrightarrow \delta^*(q_0, x) = \delta^*(q_0, y)$$

Example 1.13

Given $L \subseteq \Sigma^*$, not necessarily regular, we define R_L on Σ^*

$$xR_L y \Leftrightarrow xz \in L \Leftrightarrow yz \in L$$

Note 1.14. The index of an equivalence relation is the number of equivalence classes. Infinite equivalency classes means infinite index.

Theorem 1.15 (Myhill-nerode 1957)
The following are equivalent

1. L is regular
2. L is the union of some of the equivalence classes of a right-invariant equivalence relation R of finite index.
3. Any equivalence relation R with the properties described in (2) has to refine R_L.

Note 1.16. To say that an equivalence relation refines another means that it partitions all the equivalence relations.

Full proof is in the notes.

§1.3 Brozozowski's Algorithm

Will convert some DFA to NFA.

1. Reverse all the arrows change the initial states to an accept state, and make all the accept states start states.
2. Determinize it. This may cause exponential blow up because you are looking at the set of all subsets. Convert it from NFA to DFA.
3. Remove unreachable states.
4. Repeat 1.
5. Repeat 2. Might expect double exponential blow up, but instead there is a huge collapse to the minimal version of the original machine.
6. Repeat 3.

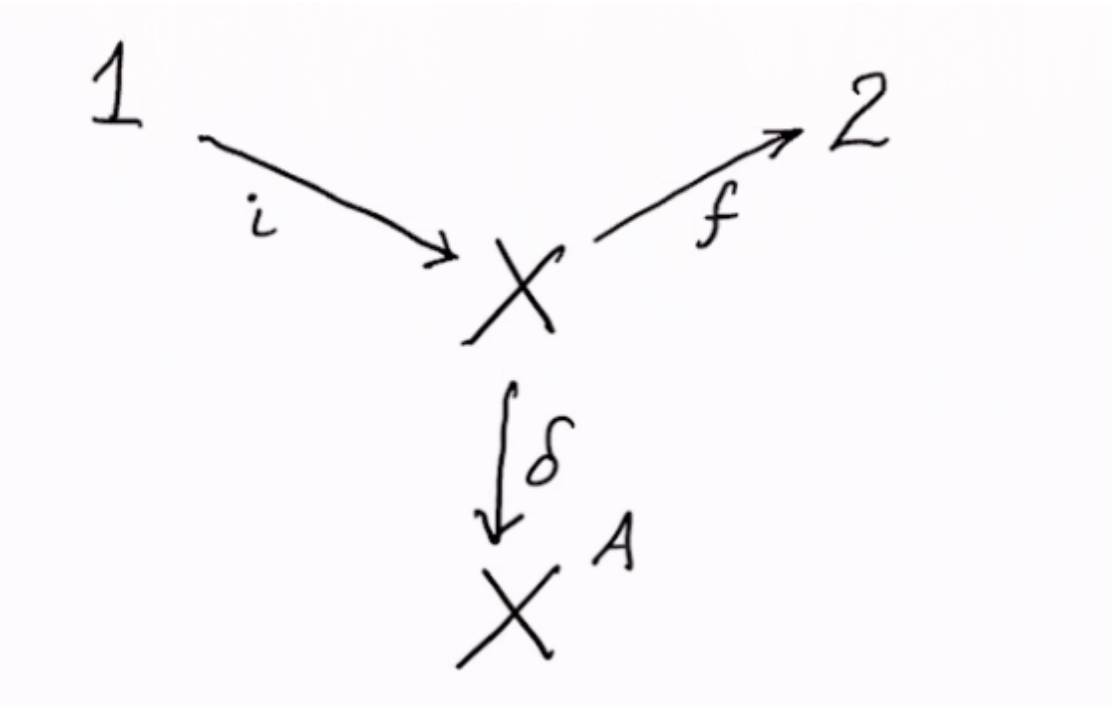
The result is the minimal machine.

Can't explain the whole thing. The point is to get interested and excited about this topic.

Note 1.17. X^A means the set of all functions from A to X. $\delta(x) : X \to X^A$.

$$X^A \approx [A \to X]$$

We can either write $\delta(x)(a)$ in which case $\delta : X \to X^A$ or $\delta(x, a)$ where $\delta : X \times A \to X$. This process is called currying.



§1.4 Infinite state automata

A^* is set of all words. Every word is a state. Initiale state is empty word.

States are words. Transition is stick the new letter at the end of the word.

There exists a unique map $r : A^* \to X$ such that the diagram above commutes.

Whatever path you follow that leads to the same end state results in the same outcome. Same as moving a different path with different endpoints.

Note 1.18. Give $f : V \to W$, $f^A : V^A \to W^A$. $\varphi \in V^A$, then $f^A(\varphi)(a) = f(\varphi(v))$. This is countably infinite.

Definition 1.19 (Reachability)**.** r maps the empty string to the . r tracks every word through the automaton with δ^*. r is the reachability map. It tells you which states can be reached. An automaton is reachable if every state can be reached.