

* Recap: We've built DFAs that recognise the languages
 ① \emptyset , ② $\{\epsilon\}$, and ③ $\{a\}$ for any $a \in \Sigma$

Next, we're trying to build a DFA M such that
 $L(M) = L(r_1 | r_2) = L(r_1) \cup L(r_2)$ given DFAs M_1 & M_2 such that
 $L(M_1) = L(r_1)$, $L(M_2) = L(r_2)$

Idea: Construct M to simulate both M_1 & M_2 on any given input, and to accept if any one accepts.

Let:

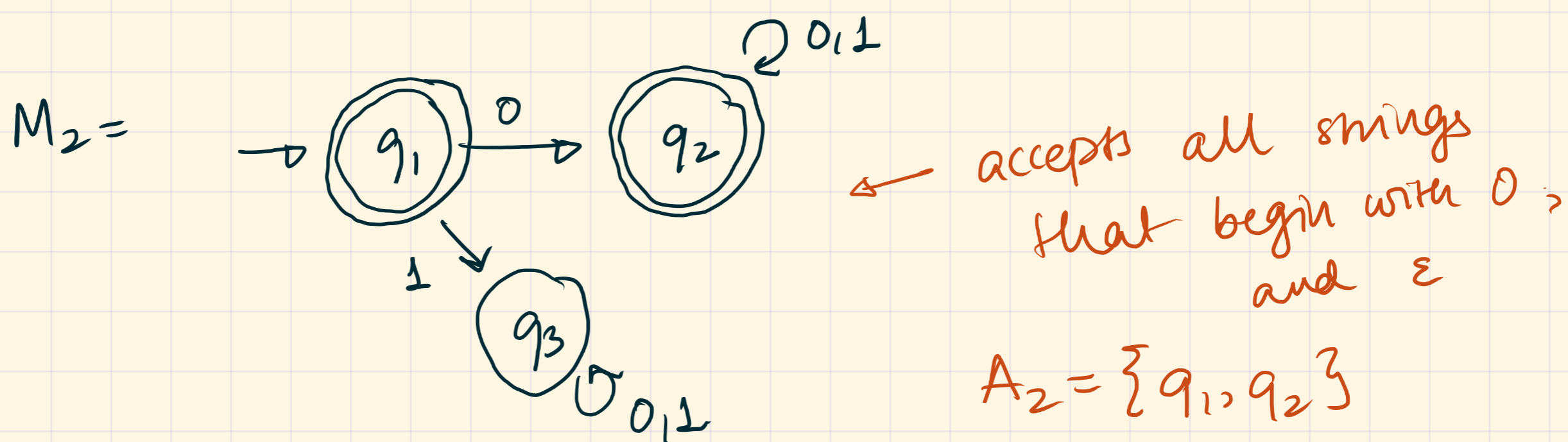
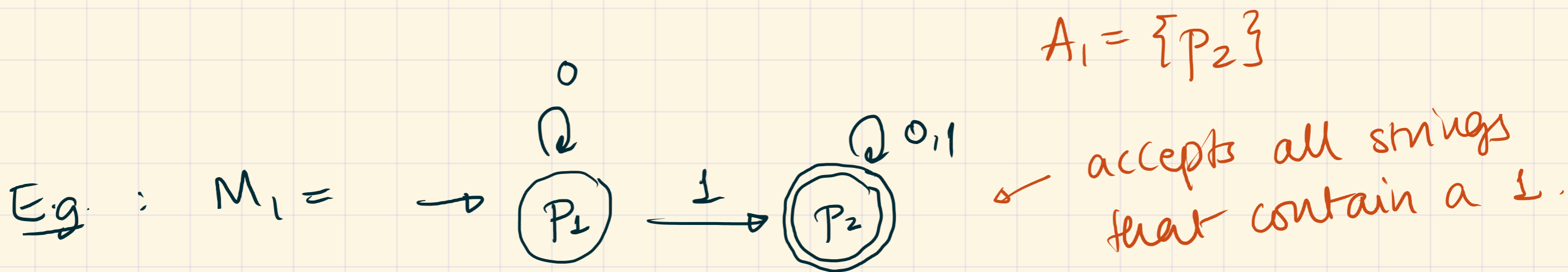
* P, Q be the states of M_1, M_2 resp.

* p_1, q_1 be the start states of M_1, M_2 resp.

* A_1, A_2 be the sets of accept states of M_1, M_2 resp

* $\delta_1: P \times \Sigma \rightarrow P$ transition fns of M_1, M_2 resp.

$\delta_2: Q \times \Sigma \rightarrow Q$



Construct M as the "product automaton".

* The states of M : $P \times Q$.

The state (p, q) represents that we're at p in M_1 , and q in M_2 .

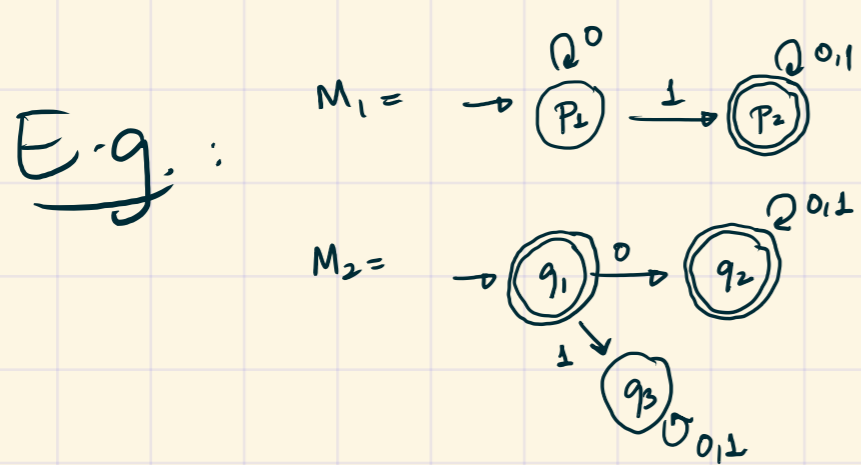
* Start state: (p_1, q_1) , representing that we start both in M_1 & in M_2 .

* Transition function:

$$\delta: (P \times Q) \times \Sigma \rightarrow (P \times Q)$$

$$(p, q), a \mapsto (\delta_1(p, a), \delta_2(q, a)).$$

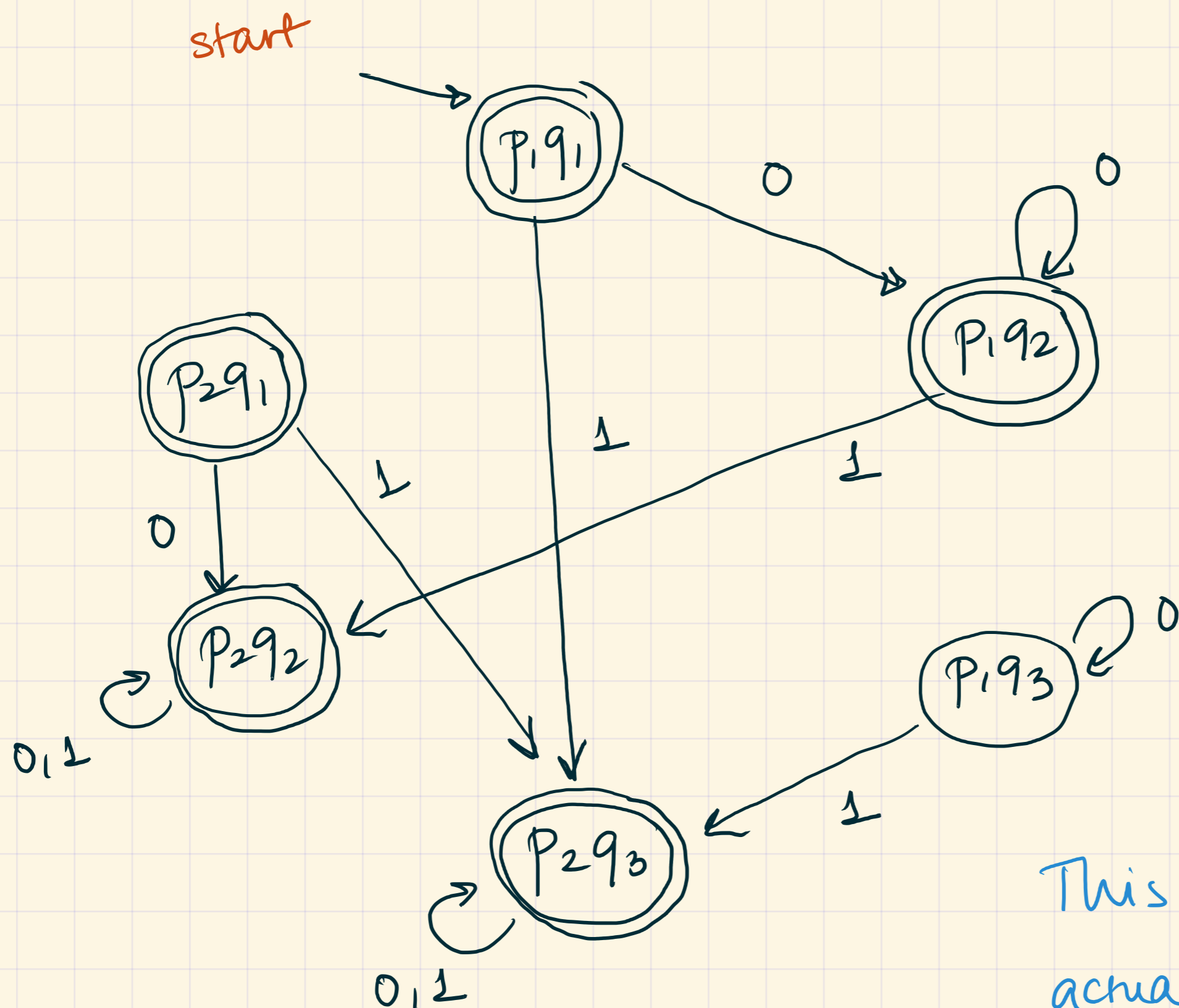
* Accept states:



We have 6 states in M :

$(p_1, q_1), (p_1, q_2), (p_1, q_3)$

$(p_2, q_1), (p_2, q_2), (p_2, q_3)$



For transitions, follow M_1 for the first coordinate and M_2 for the second coordinate.

(p, q) is accepting if either $p \in A_1$ or $q \in A_2$.

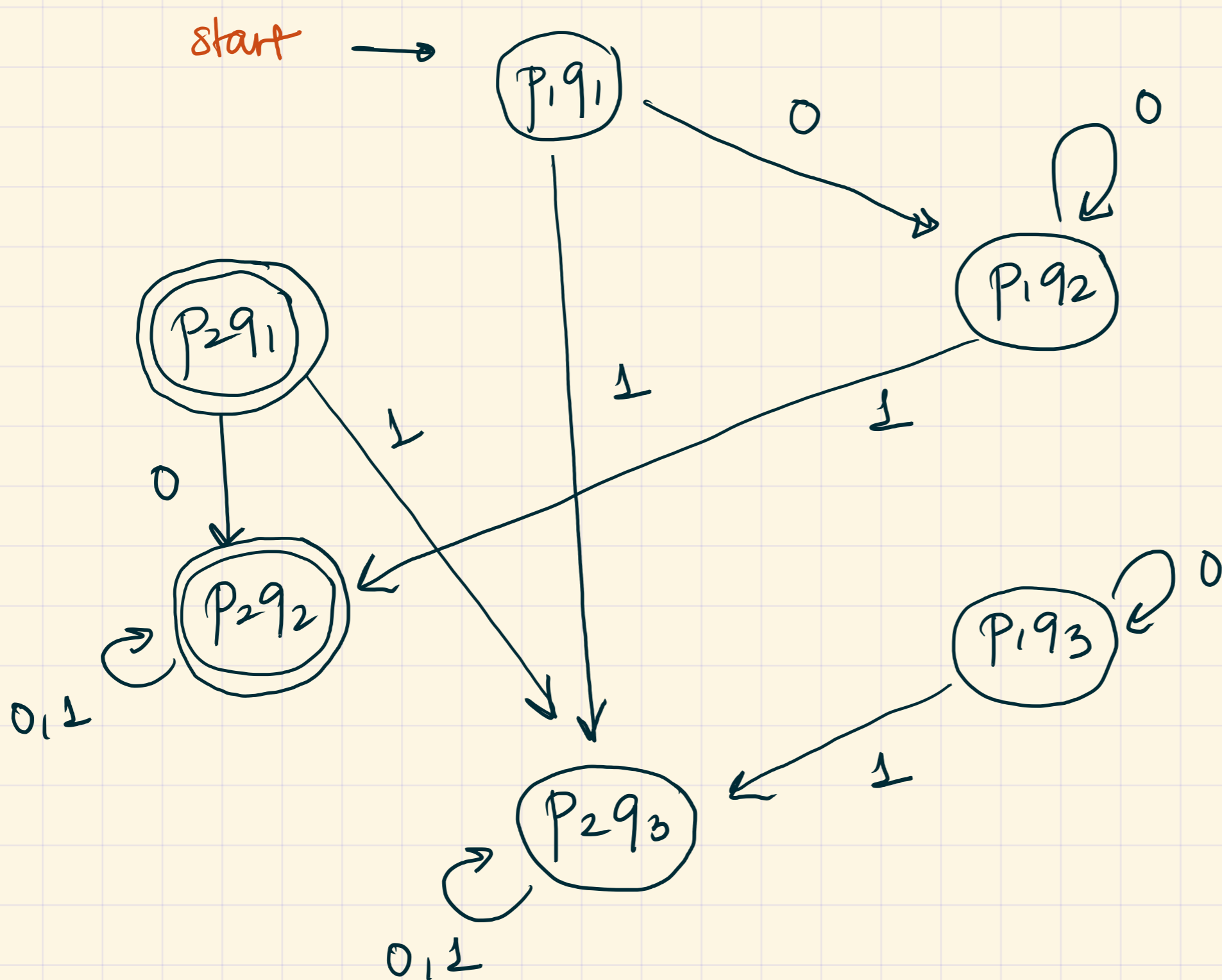
This machine M actually accepts everything!

* M will exactly accept $L(M_1) \cup L(M_2)$, but may not be the efficient machine to do so.

Here is another machine that accepts every string:



Here is something else you can do with a product automaton: you can change the accepting states to produce a machine M' that accepts $L(M_1) \cap L(M_2)$ [In example: strings that begin with a 0 and contain at least one 1]



(p, q) is accepting if $p \in A_1$ and $q \in A_2$, otherwise it's the same product construction.

Given $\gamma = \gamma_1 | \gamma_2$ & DFAs M_1, M_2 st $L(M_i) = L(\gamma_i)$,

we have now constructed M , such that $L(M) = L(\gamma)$.

(4th constructor for regex.)

The remaining two are: (5) concatenation: $\gamma = \gamma_1 \gamma_2$, and

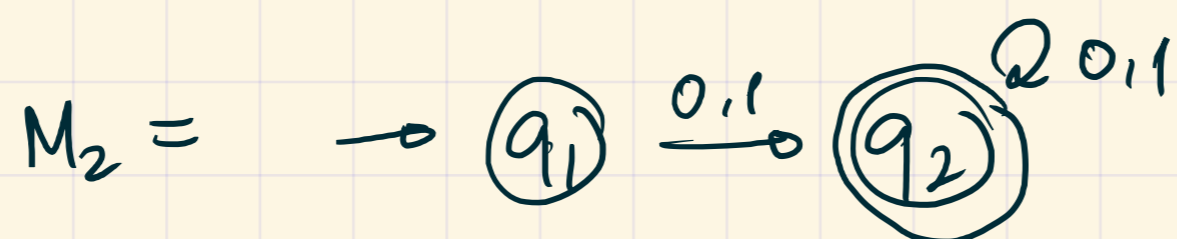
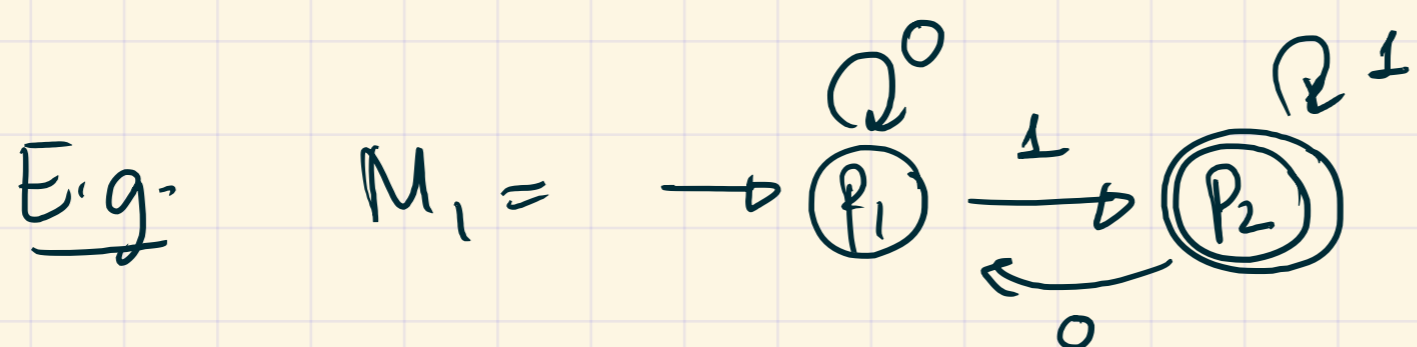
(6) star: $\gamma = (\gamma_1)^*$

⑤ : Assume we have M_1, M_2 such that $L(r_1) = L(M_1)$
 Want to build M , such that $L(M) = L(r_1, r_2)$.

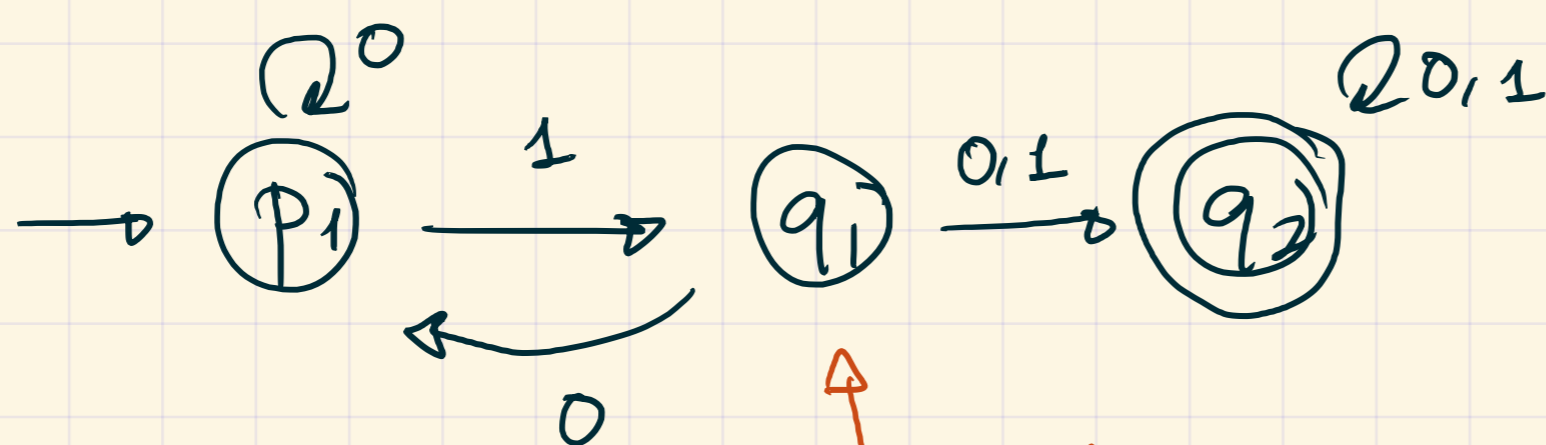
Problem: Given $w \in \Sigma^*$ that matches r_1, r_2 ,
 we know that $w = xy$ such that $x \in L(r_1), y \in L(r_2)$.
But we don't know the lengths of x & y , and in
 fact, there could be more than one way to break w !

Idea: Take $M_1 \rightarrow \text{state} \rightarrow \text{state}$ & for each accepting
 state, we'd like to
 attach a copy of M_2 .

Problem: How do you deal with the outgoing arrows from
 the accept states of M_1 ?



Suggested procedure produces:



Problem: Two outgoing arrows labelled 0!

* Next week: We'll use non-deterministic finite automata
 (NFAs) to solve this problem!