Sept, 20 th

Regular language

Definition 2.1 (Regular languages)**.** Let us fix an alphabet Σ. Then, a language L is regular iff:

1. either $L = \emptyset$;
2. or $L = \{\varepsilon\}$;
3. or $L = \{a\}$ for some $a \in \Sigma$;
4. or $L = L_1 \cup L_2$;
5. or $L = L_1 \cdot L_2$;
6. or $L = L_1^*$

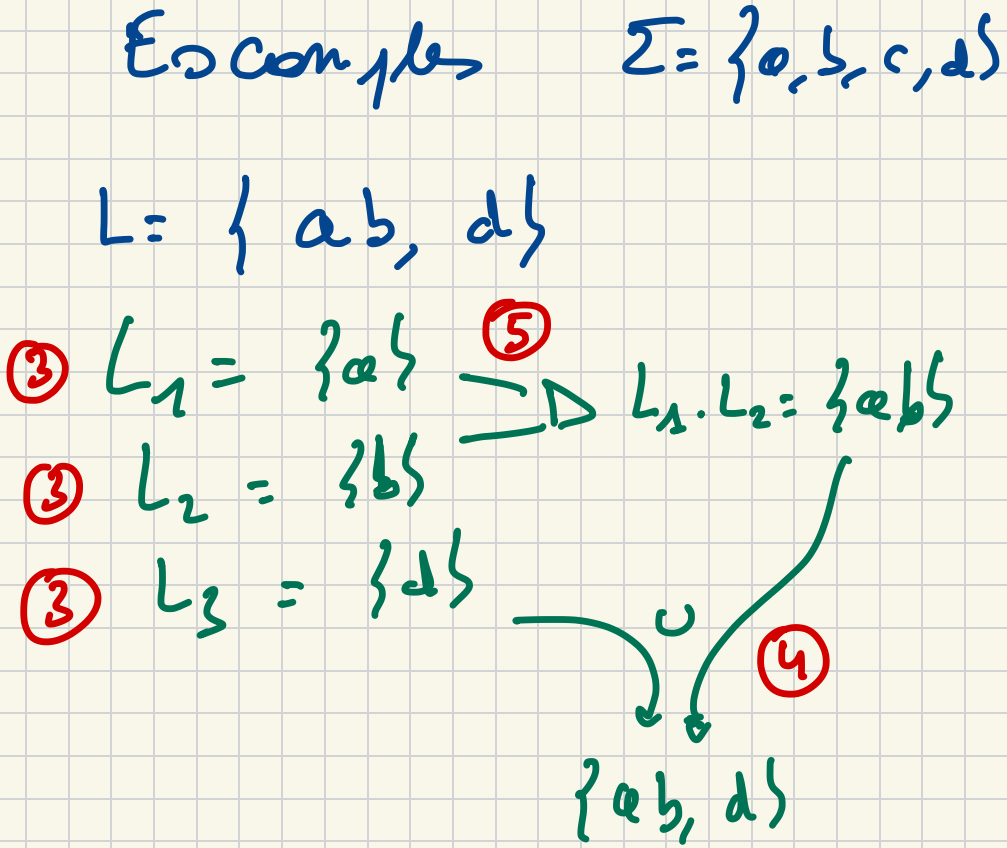
where L_1 and L_2 are regular languages on Σ.

Base cases.

Inductive cases.

⚠ finite number of times.

1. either $L = \emptyset$;
2. or $L = \{\varepsilon\}$;
3. or $L = \{a\}$ for some $a \in \Sigma$;
4. or $L = L_1 \cup L_2$;
5. or $L = L_1 \cdot L_2$;
6. or $L = L_1^*$



Using the same idea, we can prove that all finite languages are regular.

If my language is $L = \{w_1, w_2, \ldots, w_n\}$

For each word w_i, I build a language

$L_i = \{w_i\}$ rule ③ and ⑤

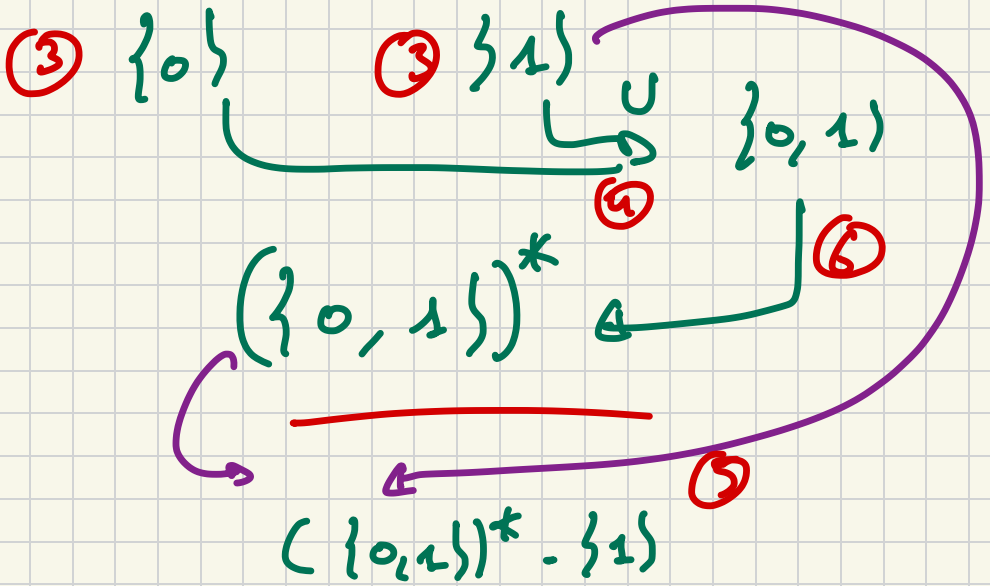
Then $L = \bigcup L_i$ ④

All binary words that represent an odd number

$\Sigma = \{0, 1\}$

"as many 0, 1 as I want"

1. either $L = \emptyset$;
2. or $L = \{\varepsilon\}$;
3. or $L = \{a\}$ for some $a \in \Sigma$;
4. or $L = L_1 \cup L_2$;
5. or $L = L_1 \cdot L_2$;
6. or $L = L_1^*$



a binary word that starts with only 0's and ends with only 1's

$$(\{0,1\})^k$$

$$\{0\}^k \cdot \{1\}^k \quad \checkmark$$

$\Sigma = \{(,)\}$ Language of all well-parenthesized words $L_()$

not regular!

$\{(,)\}^* \supset L_()$

1. either $L = \emptyset$;
2. or $L = \{\varepsilon\}$;
3. or $L = \{a\}$ for some $a \in \Sigma$;
4. or $L = L_1 \cup L_2$;
5. or $L = L_1 \cdot L_2$;
6. or $L = L_1^*$

2 kinds of tools for regular languages

- tool to define a regular language.
- tools to manipulate and recognize these languages.

We need regular language for scanning

```
int i = 5 ;

int f ( int j ) {
    int i = j ;
    return i + 1 ;
}

int main ( ) {
    printf ( "Hello␣World␣!" ) ;
    printf ( "%d␣%d" , i , f ( i + 1 ) ) ;
    return 0 ;
}
```

Definition 2.3 (Regular expressions)**.** Given a finite alphabet Σ, the following are regular expressions on Σ:

1. The constant $\emptyset$. It denotes the language $L(\emptyset) = \emptyset$.
2. The constant ε. It denotes the language $L(\varepsilon) = \{\varepsilon\}$.
3. All constants $a \in \Sigma$. Each constant $a \in \Sigma$ denotes the language $L(a) = \{a\}$.
4. All expressions of the form $r_1 + r_2$, where r_1 and r_2 are regular expressions on Σ. Each expression $r_1 + r_2$ denotes the language $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.
5. All expressions of the form $r_1 \cdot r_2$, where r_1 and r_2 are regular expressions on Σ. Each expression $r_1 \cdot r_2$ denotes the language $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$.
6. All expressions of the form r^*, where r is a regular expression on Σ. Each expression r^* denotes the language $L(r^*) = (L(r))^*$.

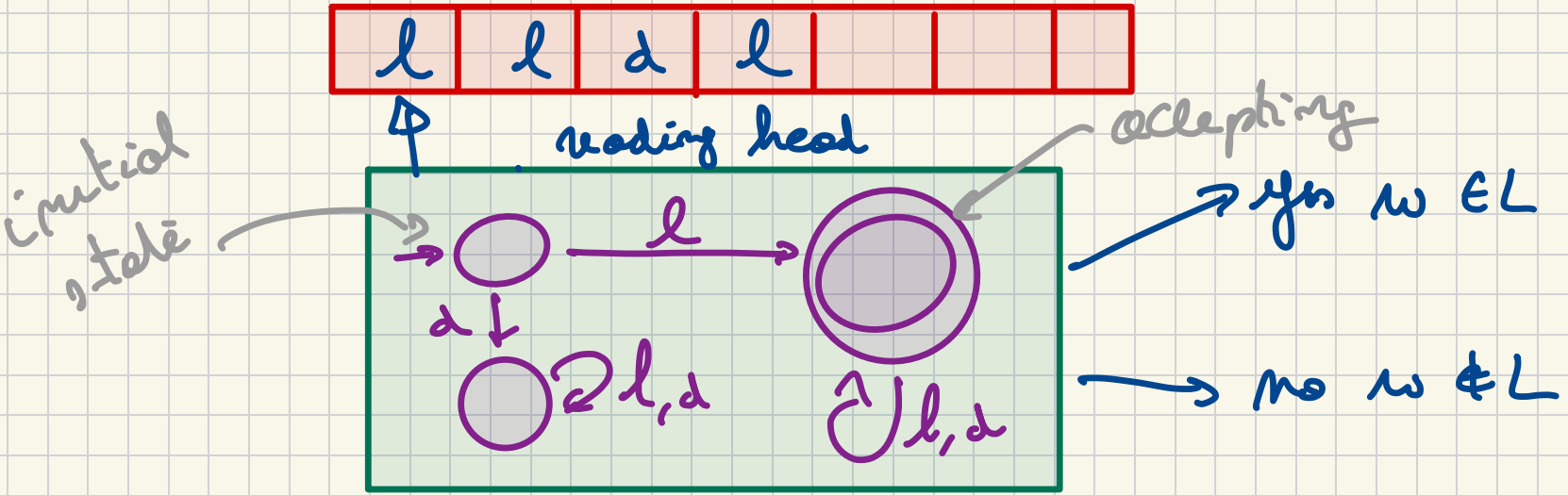
In addition, parenthesis are allowed in regular expressions to group subexpressions (with their usual semantics).

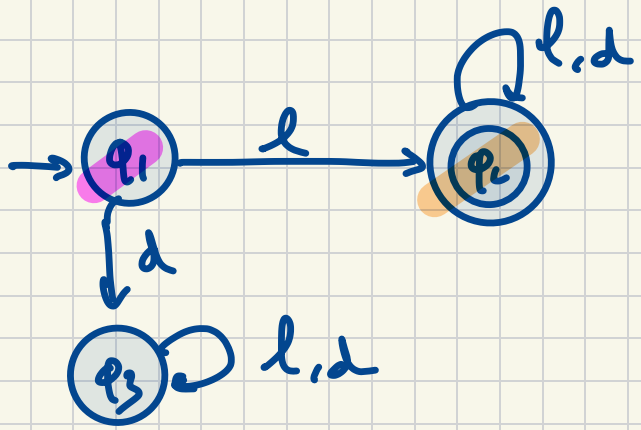
A regular expression defines a regular language.

$(0+1)^* \cdot 1$

Finite Automata

$w \in L \hookrightarrow$ regular





$l \cdot (l + d)^*$

$Q = \{q_1, q_2, q_3\}$

$\Sigma = \{l, d\}$

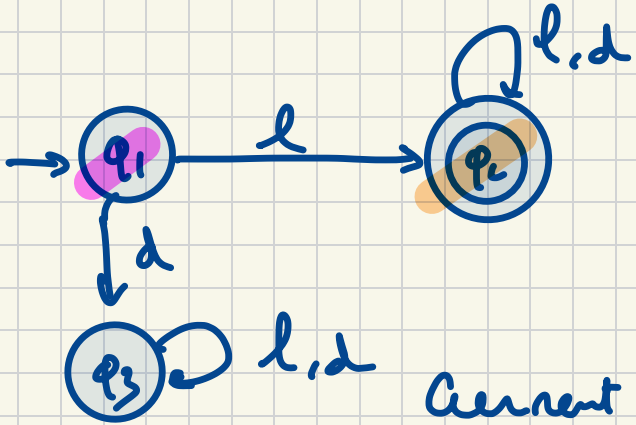
$q_0 = q_1$

$F = \{q_2\}$

$$A = \langle Q, \Sigma, \delta, q_0, F \rangle$$

where:

1. Q is a finite set of states;
2. Σ is the (finite) input alphabet;
3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \mapsto 2^Q$ is the transition function;
4. $q_0 \in Q$ is the initial state;
5. $F \subseteq Q$ is the set of accepting states.



3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the transition function;

a set of states

current state

letter in Σ which is read.

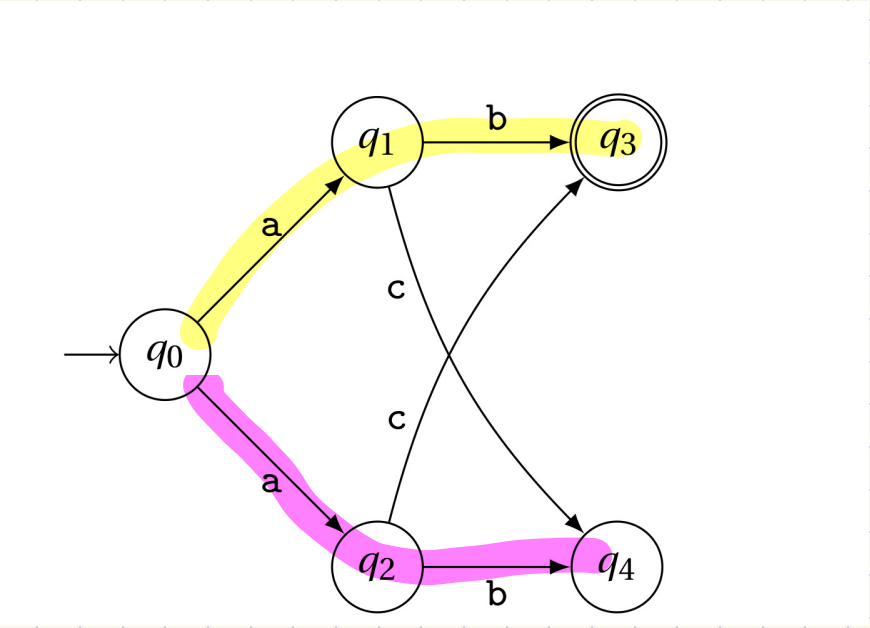
$\delta(q_1, a) = \{q_2\}$

$\delta(q_1, b) = \{q_3\}$

$\delta(q_3, a) = \{q_3\}$

$\vdots$

Non-deterministic automaton



On ab

q_0, q_1, q_3 accepts

q_0, q_2, q_4 rejects

Convention: a word is accepted if there exists at least one accepting path.

Why ???

I want an automaton that accepts all binary words where there are two 1 separated by 2 symbols