Some questions

(a) Is there an algorithm which, given a string $u$ and a regular expression $r$, computes whether or not $u$ matches $r$?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions $r$ and $s$, computes whether or not they are equivalent, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of $\Sigma^*$) of the form $L(r)$ for some $r$?
Regular languages are closed under complementation

**Lemma.** If $L$ is a regular language over alphabet $\Sigma$, then its complement \( \{u \in \Sigma^* \mid u \notin L\} \) is also regular.

**Proof.** Since $L$ is regular, by definition there is a DFA $M$ such that $L = L(M)$. Let $\text{Not}(M)$ be the DFA constructed from $M$ as indicated on Slide 92. Then $\{u \in \Sigma^* \mid u \notin L\}$ is the set of strings accepted by $\text{Not}(M)$ and hence is regular. \[\square\]

[N.B. If one applies the construction on Slide 92 (interchanging the role of accepting & non-accepting states) to a *non-deterministic* finite automaton $N$, then in general $L(\text{Not}(N))$ is not equal to $\{u \in \Sigma^* \mid u \notin L(N)\}$ – see Exercise 4.5.]

We saw on slide 79 that part (a) of Kleene’s Theorem allows us to answer question (a) on Slide 38. Now that we have proved the other half of the theorem, we can say more about question (b) on that slide. In particular, it is a consequence of Kleene’s Theorem plus the above lemma that for each regular expression $r$ over an alphabet $\Sigma$, there is a regular expression $\sim r$ that determines via matching the complement of the language determined by $r$:

$$L(\sim r) = \{u \in \Sigma^* \mid u \notin L(r)\}$$

To see this, given a regular expression $r$, by part (a) of Kleene’s Theorem there is a DFA $M$ such that $L(r) = L(M)$. Then by part (b) of the theorem applied to the DFA $\text{Not}(M)$, we can find a regular expression $\sim r$ so that $L(\sim r) = L(\text{Not}(M))$. Since $L(\text{Not}(M)) = \{u \in \Sigma^* \mid u \notin L(M)\} = \{u \in \Sigma^* \mid u \notin L(r)\}$, this $\sim r$ is the regular expression we need for the complement of $r$. 

91
**Not(M)**

Given DFA \( M = (Q, \Sigma, \delta, s, F) \), then \( \text{Not}(M) \) is the DFA with

- set of states = \( Q \)
- input alphabet = \( \Sigma \)
- next-state function = \( \delta \)
- start state = \( s \)
- accepting states = \( \{ q \in Q \mid q \notin F \} \).

(i.e. we just reverse the role of accepting/non-accepting and leave everything else the same)

Because \( M \) is a deterministic finite automaton, then \( u \) is accepted by \( \text{Not}(M) \) iff it is not accepted by \( M \):

\[
L(\text{Not}(M)) = \{ u \in \Sigma^* \mid u \notin L(M) \}
\]
**Theorem.** If $L_1$ and $L_2$ are regular languages over an alphabet $\Sigma$, then their intersection $L_1 \cap L_2 = \{ u \in \Sigma^* \mid u \in L_1 \& u \in L_2 \}$ is also regular.

**Proof.** Note that $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2))$

(cf. de Morgan’s Law: $p \& q = \neg(\neg p \lor \neg q)$).

So if $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for DFAs $M_1$ and $M_2$, then $L_1 \cap L_2 = L(\text{Not}(\text{PM}))$ where $M$ is the NFA$^\varepsilon$ $\text{Union}(\text{Not}(M_1), \text{Not}(M_2))$.

[It is not hard to directly construct a DFA $\text{And}(M_1, M_2)$ from $M_1$ and $M_2$ such that $L(\text{And}(M_1, M_2)) = L(M_1) \cap L(M_2)$ – see Exercise 4.7.]
Regular languages are closed under intersection

**Corollary:** given regular expressions $r_1$ and $r_2$, there is a regular expression, which we write as $r_1 \& r_2$, such that a string $u$ matches $r_1 \& r_2$ iff it matches both $r_1$ and $r_2$.

**Proof.** By Kleene (a), $L(r_1)$ and $L(r_2)$ are regular languages and hence by the theorem, so is $L(r_1) \cap L(r_2)$. Then we can use Kleene (b) to construct a regular expression $r_1 \& r_2$ with $L(r_1 \& r_2) = L(r_1) \cap L(r_2)$. □
Some questions

(a) Is there an algorithm which, given a string $u$ and a regular expression $r$, computes whether or not $u$ matches $r$?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions $r$ and $s$, computes whether or not they are equivalent, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of $\Sigma^*$) of the form $L(r)$ for some $r$?
Definition. Two regular expressions $r$ and $s$ are said to be equivalent if $L(r) = L(s)$, that is, they determine exactly the same sets of strings via matching.

For example, are $b^*a(b^*a)^*$ and $(a|b)^*a$ equivalent?

Answer: yes (Exercise 2.3)

How can we decide all such questions?
Note that $L(r) = L(s)$

iff $L(r) \subseteq L(s)$ and $L(s) \subseteq L(r)$

iff $(\Sigma^* \setminus L(r)) \cap L(s) = \emptyset = (\Sigma^* \setminus L(s)) \cap L(r)$

iff $L((\sim r) \& s) = \emptyset = L((\sim s) \& r)$

iff $L(M) = \emptyset = L(N)$

where $M$ and $N$ are DFAs accepting the sets of strings matched by the regular expressions $(\sim r) \& s$ and $(\sim s) \& r$ respectively.

So to decide equivalence for regular expressions it suffices to check, given any given DFA $M$, whether or not it accepts some string.

Note that the number of transitions needed to reach an accepting state in a finite automaton is bounded by the number of states (we can remove loops from longer paths). So we only have to check finitely many strings to see whether or not $L(M)$ is empty.
The Pumping Lemma
Some questions

(a) Is there an algorithm which, given a string $u$ and a regular expression $r$, computes whether or not $u$ matches $r$?

(b) In formulating the definition of regular expressions, have we missed out some practically useful notions of pattern?

(c) Is there an algorithm which, given two regular expressions $r$ and $s$, computes whether or not they are equivalent, in the sense that $L(r)$ and $L(s)$ are equal sets?

(d) Is every language (subset of $\Sigma^*$) of the form $L(r)$ for some $r$?
In the context of programming languages, a typical example of a regular language (Slide 64) is the set of all strings of characters which are well-formed *tokens* (basic keywords, identifiers, etc) in a particular programming language, Java say. By contrast, the set of all strings which represent well-formed Java *programs* is a typical example of a language that is not regular. Slide 101 gives some simpler examples of non-regular languages. For example, there is no way to use a search based on matching a regular expression to find all the palindromes in a piece of text (although of course there are other kinds of algorithm for doing this).

The intuitive reason why the languages listed on Slide 101 are not regular is that a machine for recognising whether or not any given string is in the language would need *infinitely* many different states (whereas a characteristic feature of the machines we have been using is that they have only *finitely* many states). For example, to recognise that a string is of the form $a^n b^n$ one would need to remember how many $a$s had been seen before the first $b$ is encountered, requiring countably many states of the form ‘just_seen_n_a$s’. This section make this intuitive argument rigorous and describes a useful way of showing that languages such as these are not regular.

The fact that a finite automaton does only have finitely many states means that as we look at longer and longer strings that it accepts, we see a certain kind of repetition—the **pumping lemma property** given on Slide 102.
Examples of languages that are not regular

- The set of strings over \( \{ (, ), a, b, \ldots, z \} \) in which the parentheses ‘(’ and ‘)’ occur well-nested.
- The set of strings over \( \{ a, b, \ldots, z \} \) which are palindromes, i.e. which read the same backwards as forwards.
- \( \{ a^n b^n \mid n \geq 0 \} \)
The Pumping Lemma

For every regular language $L$, there is a number $\ell \geq 1$ satisfying the **pumping lemma property**:

All $w \in L$ with $|w| \geq \ell$ can be expressed as a concatenation of three strings, $w = u_1v u_2$, where $u_1$, $v$ and $u_2$ satisfy:

- $|v| \geq 1$ (i.e. $v \neq \varepsilon$)
- $|u_1v| \leq \ell$
- for all $n \geq 0$, $u_1v^n u_2 \in L$ (i.e. $u_1u_2 \in L$, $u_1v u_2 \in L$ [but we knew that anyway], $u_1vv u_2 \in L$, $u_1vvv u_2 \in L$, etc.)
Proving the Pumping Lemma

Since $L$ is regular, it is equal to the set $L(M)$ of strings accepted by some DFA $M = (Q, \Sigma, \delta, s, F)$. Then we can take the number $\ell$ mentioned on Slide 102 to be the number of states in $Q$. For suppose $w = a_1a_2\ldots a_n$ with $n \geq \ell$. If $w \in L(M)$, then there is a transition sequence as shown at the top of Slide 104. Then $w$ can be split into three pieces as shown on that slide. Note that by choice of $i$ and $j$, $|v| = j - i \geq 1$ and $|u_1v| = j \leq \ell$. So it just remains to check that $u_1v^nu_2 \in L$ for all $n \geq 0$. As shown on the lower half of Slide 104, the string $v$ takes the machine $M$ from state $q_i$ back to the same state (since $q_i = q_j$). So for any $n$, $u_1v^nu_2$ takes us from the initial state $s_M = q_0$ to $q_i$, then $n$ times round the loop from $q_i$ to itself, and then from $q_i$ to $q_n \in \text{Accept}_M$. Therefore for any $n \geq 0$, $u_1v^nu_2$ is accepted by $M$, i.e. $u_1v^nu_2 \in L$. □

[Note. In the above construction it is perfectly possible that $i = 0$, in which case $u_1$ is the null-string, $\varepsilon$.]

Remark. One consequence of the pumping lemma property of $L$ and $\ell$ is that if there is any string $w$ in $L$ of length $\geq \ell$, then $L$ contains arbitrarily long strings. (We just ‘pump up’ $w$ by increasing $n$.) If you did Exercise 4.3, you will know that if $L$ is a finite set of strings then it is regular. In this case, what is the number $\ell$ with the property on Slide 102? The answer is that we can take any $\ell$ strictly greater than the length of any string in the finite set $L$. Then the Pumping Lemma property is trivially satisfied because there are no $w \in L$ with $|w| \geq \ell$ for which we have to check the condition!
Suppose $L = L(M)$ for a DFA $M = (Q, \Sigma, \delta, s, F)$. Taking $\ell$ to be the number of elements in $Q$, if $\ell \geq n$, then in

$$s = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_\ell} q_\ell \cdots \xrightarrow{a_n} q_n \in F$$

$\ell+1$ states

$q_0, \ldots, q_\ell$ can’t all be distinct states. So $q_i = q_j$ for some $0 \leq i < j \leq \ell$. So the above transition sequence looks like

$$s = q_0 \xrightarrow{u_1} q_i = q_j \xrightarrow{v} q_n \in F$$

where

$$u_1 \triangleq a_1 \ldots a_i \quad v \triangleq a_{i+1} \ldots a_j \quad u_2 \triangleq a_{j+1} \ldots a_n$$
How to use the Pumping Lemma to prove that a language $L$ is not regular

For each $\ell \geq 1$, find some $w \in L$ of length $\geq \ell$ so that no matter how $w$ is split into three, $w = u_1vru_2$, with $|u_1v| \leq \ell$ and $|v| \geq 1$, there is some $n \geq 0$ for which $u_1v^n u_2$ is not in $L$.

(†)
Examples

None of the following three languages are regular:

(i) \( L_1 \triangleq \{ a^n b^n \mid n \geq 0 \} \)
    
    [For each \( \ell \geq 1 \), \( a^\ell b^\ell \in L_1 \) is of length \( \geq \ell \) and has property (†) on Slide 105.]

(ii) \( L_2 \triangleq \{ w \in \{a, b\}^* \mid w \text{ a palindrome} \} \)
    
    [For each \( \ell \geq 1 \), \( a^\ell ba^\ell \in L_1 \) is of length \( \geq \ell \) and has property (†).]

(iii) \( L_3 \triangleq \{ a^p \mid p \text{ prime} \} \)
    
    [For each \( \ell \geq 1 \), we can find a prime \( p \) with \( p > 2\ell \) and then \( a^p \in L_3 \) has length \( \geq \ell \) and has property (†).]
Using the Pumping Lemma

The Pumping Lemma (Slide 102) says that every regular language has a certain property – namely that there exists a number \( \ell \) with the pumping lemma property. So to show that a language \( L \) is not regular, it suffices to show that no \( \ell \geq 1 \) possesses the pumping lemma property for the language \( L \). Because the pumping lemma property involves quite a complicated alternation of quantifiers, it will help to spell out explicitly what is its negation. This is done on Slide 105.

Slide 106 gives some examples:

(i) For any \( \ell \geq 1 \), consider the string \( w = a^{\ell}b^{\ell} \). It is in \( L_1 \) and has length \( \geq \ell \). We show that property (\( \dagger \)) holds for this \( w \). For suppose \( w = a^{\ell}b^{\ell} \) is split as \( w = u_1v u_2 \) with \( |u_1v| \leq \ell \) and \( |v| \geq 1 \). Then \( u_1v \) must consist entirely of \( a \)s, so \( u_1 = a^r \) and \( v = a^s \) say, and hence \( u_2 = a^{\ell-r-s}b^{\ell} \). Then the case \( n = 0 \) of \( u_1v^n u_2 \) is not in \( L_1 \) since

\[
u_1v^0 u_2 = u_1 u_2 = a^r (a^{\ell-r-s}b^{\ell}) = a^{\ell-s}b^{\ell}\]

and \( a^{\ell-s}b^{\ell} \notin L_1 \) because \( \ell - s \neq \ell \) (since \( s = |v| \geq 1 \)).

(ii) The argument is very similar to that for example (i), but starting with the palindrome \( w = a^{\ell}ba^{\ell} \). Once again, the \( n = 0 \) case of \( u_1v^n u_2 \) yields a string \( u_1 u_2 = a^{\ell-s}ba^{\ell} \) which is not a palindrome (because \( \ell - s \neq \ell \)).
(iii) Given $\ell \geq 1$, since [Euclid proved that] there are infinitely many primes $p$, we can certainly find one satisfying $p > 2\ell$. I claim that $w = a^p$ has property (†). For suppose $w = a^p$ is split as $w = u_1v u_2$ with $|u_1v| \leq \ell$ and $|v| \geq 1$. Letting $r \triangleq |u_1|$ and $s \triangleq |v|$, so that $|u_2| = p - r - s$, we have

$$u_1v^{p-s}u_2 = a^r a^s(p-s) a^{p-r-s} = a^{sp-s^2+p-s} = a^{(s+1)(p-s)}$$

Now $(s + 1)(p - s)$ is not prime, because $s + 1 > 1$ (since $s = |v| \geq 1$) and $p - s > 2\ell - \ell = \ell \geq 1$ (since $p > 2\ell$ by choice, and $s \leq r + s = |u_1v| \leq \ell$). Therefore $u_1v^n u_2 \notin L_3$ when $n = p - s$.

**Remark.** Unfortunately, the method on Slide 105 cannot cope with every non-regular language. This is because the pumping lemma property is a necessary, but not a sufficient condition for a language to be regular. In other words there do exist languages $L$ for which a number $\ell \geq 1$ can be found satisfying the pumping lemma property on Slide 102, but which nonetheless, are not regular. Slide 109 gives an example of such an $L$. 
Example of a non-regular language with the pumping lemma property

\[ L \triangleq \{ c^m a^n b^n \mid m \geq 1 \& n \geq 0 \} \cup \{ a^m b^n \mid m, n \geq 0 \} \]

satisfies the pumping lemma property on Slide 102 with \( \ell = 1 \).

[For any \( w \in L \) of length \( \geq 1 \), can take \( u_1 = \varepsilon \), \( v = \) first letter of \( w \), \( u_2 = \) rest of \( w \).]

But \( L \) is not regular – see Exercise 5.1.