Languages and automata

1. Regular expression
2. Deterministic finite automaton, error state
3. Nondeterministic finite automaton, nondeterministic finite automaton with ε-moves
4. You should know Kleene's theorem, and in particular, how to algorithmically convert a regular expression to a DFA. You don't need to know how to convert a DFA to a regular expression.
5. You should be able to prove that a language isn't regular. (The Pumping Lemma wasn't covered and isn't needed.)
6. Equivalence of automata.
7. Automata for complement and intersection.
8. Minimization,
9. Context free languages (not pushdown automata).
10. Chomsky Normal Form.
11. Ambiguity of context free grammars.

Complexity

1. Complexity of algorithms, complexity of problems
2. Lower and upper bounds
3. O, Ω, θ notation
4. Polynomial and exponential complexity
5. **NP** problems, the two definitions, the question of whether **P** = **NP**. (You're not required to know the answer.)
6. **NP**-completeness and SAT.

Turing machines

1. Execute Turing machines
2. Design Turing machines for simple tasks
3. Fancy Turing machines (extended alphabet, 2 tape etc.) can be simulated by an ordinary Turing machine.
4. Church's thesis: any function on words that can be computed algorithmically can be computed by a Turing machine

Decidability

1. problem, decision problem (a problem whose answer is yes/no)
2. decidable and semidecidable problems.
3. The Halting Theorem: whether a given program halts on given inputs is undecidable.
4. Rice's Theorem: any semantic property of code that sometimes hold and sometimes fails to hold is undecidable.
5. If you're asked to show a problem is decidable, write a program to decide it. Or at least sketch how you would write a program.
6. If you're asked to show a problem is undecidable, you might reduce the halting problem to it, or you might appeal to Rice's theorem.

Lambda-calculus

1. λ and application
2. Evaluating using β and δ-conversions until you reach normal form
3. Remember you can α-convert as much as you like

4. The reduction graph of a term
5. Types, and finding the principal (most general) type
6. The Church-Rosser theorem
7. Untyped terms can get stuck or run forever.

Note that at each point you should know practical aspects, e.g. uses of regular and context free languages, advantages and disadvantages of types, etc.

Suggested Solutions for 2012 Exam

1. (a) i. `(c|a)*(`ε `| b(b|c)*)`

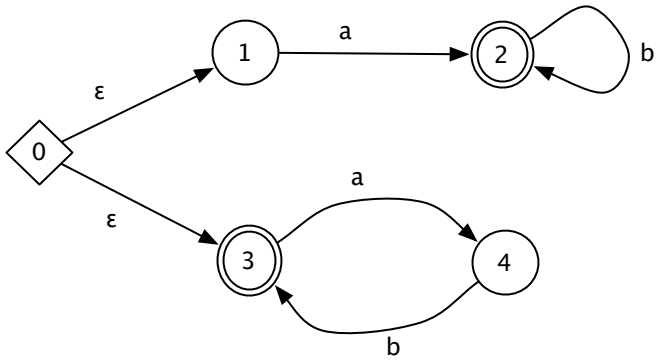
   ii. The string `ccaccbcc` consists of two parts

   - `ccacc`, which matches `(c|a)*`
   - `bcc` which matches `b(b|c)*` and therefore ε `| b(b|c)*`

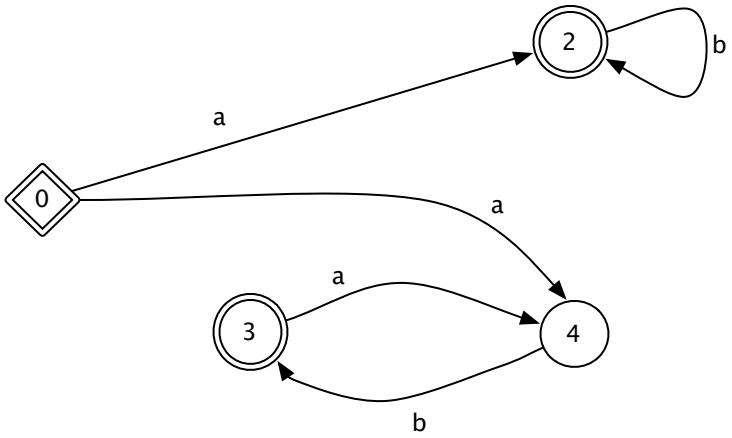
   On the other hand `ccbccacc` does not divide up in any way that matches the pattern; it the second part is to be empty, the first part must be `ccbccacc`, which is impossible, and if the first part begins with `b`, the first part will be `cc` and the second part is `bccacc` but that doesn't match `b(b|c)*`.

   (b) NB please replace every diamond by a circle marked with ⇒.

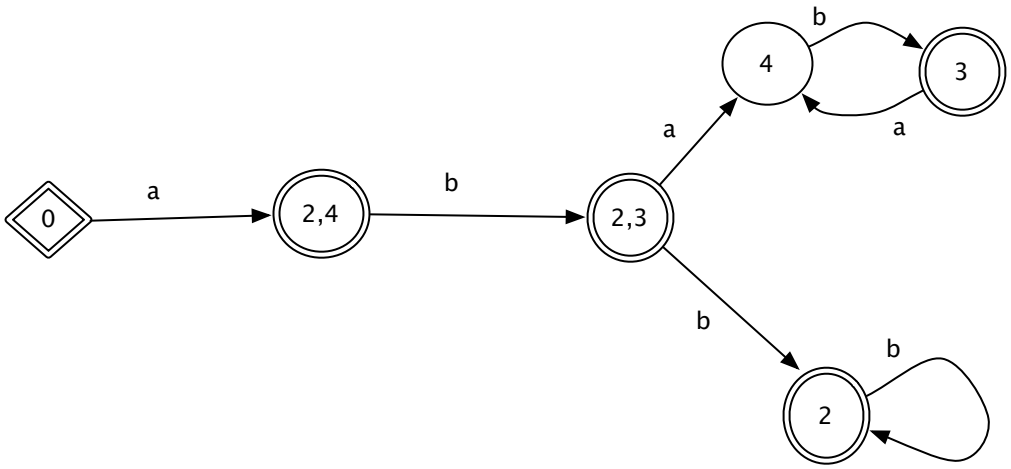
   Here is a nondeterministic finite automaton with ε-moves:



   Removing the ε-moves gives



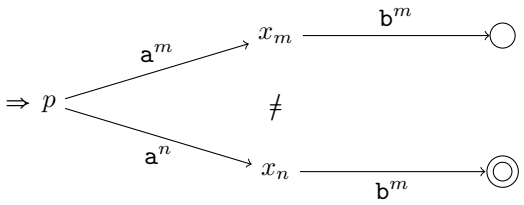
   (I have also removed state 1 since it cannot be reached.) Determinizing this gives



   (c) [The question asked for a proof using the Pumping Lemma. Here is a proof that doesn't use the Pumping Lemma.]

If L is regular then there's a DFA $(X, p, \delta, \mathsf{Acc})$ that recognizes it. Let x_n be the state we reach when we start at p and read $\mathtt{a}^n$. We shall show these states are all distinct, which implies that there are infinitely many states, a contradiction.

Suppose m and n are natural numbers with $m < n$. We want to show that $x_m \neq x_n$. If we start at x_m and read $\mathtt{b}^m$ we reach a rejecting state, because $\mathtt{a}^m\mathtt{b}^m \notin L$, but if we start at x_n and read $\mathtt{b}^m$ we reach an accepting state, because $\mathtt{a}^n\mathtt{b}^m \in L$. So x_m and x_n can't be the same.



2. (a) Suppose `xlrxrxlx` is an S-word. Since it's not empty, it has to divide into a T-word t then an S-word s. Now t doesn't begin with `l` so must be `x`. So s is `lrxrxlx`. Since it's not empty, it divides into a T-word t' then an S-word s'. Now t' isn't `x` so it must begin with `l` and end with `r`; so it's either `lrxr` or `lr`. If t' = `lrxr`, then since `lrxr` isn't `x`, we must have that `rx` is an S-word; since it isn't empty it must divide into a T-word then an S-word, but no prefix of `rx` is either `x` or begins with `l`, contradiction. So t' = `lr` and s' is `xrxlx`, and the latter divides into a T-word, clearly `x`, followed by an S-word, which must be `rxlx`. This in turn, since it's not empty, must divide into a T-word followed by an S-word, but no prefix of `rxlx` is either `x` or begins with `l`, contradiction. Therefore `xlrxrxlx` is not an S-word.

   (b)

   Not on the syllabus.

   (c) Not on the syllabus.

   (d) We shall show by induction that every S-word and also every T-word is well-bracketed.

   - ε is well-bracketed.
   - If t is well-bracketed and s is well-bracketed then ts is well-bracketed, because the bracket count at the end of t is 0 so the bracket count in the s part is the same as in s alone.
   - `x` is well-bracketed.
   - If s is well-bracketed then `l`s`r` is well-bracketed, because the bracket count after `l` is 1, so the bracket count in the s part is 1 greater than it is in s alone, so it can never go < 1 and ends at 1. The final `r` then takes the bracket count to 0 as required.

3. Note that the Turing machines we have studied this year are in a simpler format than the ones we taught in 2011-12.

   (a) i.

δ	a	b	␣
$q_0 = 0$			(1,r)
1	(2,␣)	(2,␣)	(5,n)
2			(3,r)
3	(3,r)	(3,r)	(4,l)
4	(5,␣)	(5,␣)	(5,␣)
5	stop	stop	stop

   ii.

$$
\begin{array}{l}
{}^{0}\!\text{␣abb␣} \rightarrow \\
\text{␣}^{1}\!\text{abb␣} \rightarrow \\
\text{␣␣}^{2}\!\text{bb␣} \rightarrow \\
\text{␣␣}^{3}\!\text{bb␣} \rightarrow \\
\text{␣␣b}^{3}\!\text{b␣} \rightarrow \\
\text{␣␣bb}^{3}\!\text{␣} \rightarrow \\
\text{␣␣}^{4}\!\text{bb␣} \rightarrow \\
\text{␣␣b}^{5}\!\text{␣␣} \rightarrow \\
\text{stop}
\end{array}
$$

(b) It means that there exists numbers N and C and k such that, if the length n of the input is at least N, then the machine terminates in time $\leqslant Cn^k$. We intersperse the two tapes as follows: If the main tape is

$$a_{-3}a_{-2}\overset{H}{a_{-1}}a_0a_1a_2$$

and the secondary tape is

$$b_{-1}b_0b_1b_2b_3\overset{H}{b_4}b_5$$

then the single tape simulating them is

$$\overset{H}{L}a_{-3}\sqcup a_{-2}\sqcup a'_{-1}b_{-1}a_0b_0a_1b_1a_2b_2\sqcup b_3\sqcup b'_4\sqcup b_5R$$

in the same state. (We assume an extra tape character b' for each input character b, and extra tape characters L and R.)

- To work on the main tape we first move right (once and then two at a time) until we hit the dashed character representing the head position on the main tape. Then
  - to simulate a move right, we move two steps to the right, and dash, moving the R position if necessary.
  - likewise to simulate a move left
  - to simulate an output, output the appropriate dashed character.

  Finally we move left until we reach the L position.
- Likewise to work on the secondary tape.

(c) Suppose M executes in a time polynomial in the length x of the input, so it is bounded by $\leqslant Cx^k$. Then the space used is $\leqslant x + Cx^k$. Each step in the simulation takes at most this many steps, so the overall time taken by the simulation is

$$\leqslant Cx^k \times (x + Cx^k)$$

which is polynomial of order $2k$.

(d) i. Yes, this is polynomial in the length of n. Just do long division and see if there is a remainder.

ii. Yes, this is primality testing, which is polynomial in the length of n.

iii. Not known to be polynomial.

(iii) is in $\mathcal{NP}$ since by (i) it has a checking problem polynomial in n. Therefore it is polynomial if $\mathcal{P} = \mathcal{NP}$. It is not known whether (iii) is $\mathcal{NP}$-complete, and it might be polynomial even if $\mathcal{P} \neq \mathcal{NP}$.

4. (a) Being semantic means that the input-output behaviour (the infinite table showing next to every possible inputs, the corresponding output or nontermination) determines whether the property holds or not. Rice's theorem says that every such property, if it holds in some cases and fails to hold in some cases, is undecidable.

(b) The first one is a semantic property. It is satisfied by a program that just prints the first input character, and not satisfied by a program that just hangs. Therefore by Rice's theorem it is undecidable.

As for the second, assuming the filename is supposed to be passed as an argument to the procedure, the input consists of both the filename and the contents of the file. So the property is semantic. It is satisfied by a program that just prints the first letter of the filename and not satisfied by a program that just hangs. Therefore by Rice's theorem it is undecidable.

(c) It is bracketed as $(x\,y)\,z$. This may be interpreted as applying x first to the parameter y and then to the parameter z.

(d) Not covered this year.

(e) Not covered this year.