Examples of DFA constructions, non-deterministic automata. For the following questions assume that the alphabet is \( \Sigma = \{0,1\} \) unless stated otherwise.

1. Construct deterministic finite state automata for the following languages
   - \( L_1 = \{ \, w \mid w \text{ has even number of } 1\text{s} \} \)
   - \( L_2 = \{ \, w \mid w \text{ has even number of } 0\text{s and } 1\text{s} \} \)
   - \( L_3 = \{ \, w \mid w \text{ start and ends with the same symbol} \} \)
   - \( L_4 = \{ \, w \mid w \text{ starts with a } '0' \text{ and ends with a } '1' \} \)
• L5 = \{ w | w contains '010' as a substring \}
  construct both deterministic and non-deterministic FAs for this one

*Here is a non-deterministic one:*

2. How many states you need for a DFA that recognizes L2?

3. Find a language that has small NFAs but requires large DFAs
   (What would you consider 'small' and 'large'?)

   *Hint: k-th symbol from the end is a 0. Also see below for more on this problem*

---

**Tutorial week 3**

Topics: Regular languages, finding the equivalent regular expression for a given nondeterministic automaton

1. Is the following language regular?
   L = \{ w | the number of occurrences of '01' in w is equal to the number of occurrences of '10' \}

   *Hint: Yes this language is regular although it “looks” like it is not regular since its
definition involves counting in some sense. However you can see that you do not need to
count the occurrences of 01 and 10 since it is impossible to have a second occurrence of
01 without an occurrence of 10 in between. for example 0101 has two occurrences of 01
but also one occurrence of 10 as well. so we never need to count to anything higher than
one or two. this can be done by an automaton.

2. Describe a language that has small non-deterministic automata but would require large
deterministic automata

Hint: Here is an example: the k-th symbol from the end of the automaton is 0, for some
fixed k, or the last k characters of the string are these “01101000010...001”. for example
the 10-th symbol from the end is a 0. A non-deterministic automaton can have just a
simple part that recognizes the pattern we want (0ssssssss) and non-deterministically
jump to that part from the starting state that has a loop around it to ‘eat-up’ the the first
part of the input string until we reach the k last character. so non-deterministically we
can guess that we are at the k-th character and start checking whether the input string
has the desired property. However with a deterministic automaton that is not possible.
We need to remember a window of all k last seen character because, for all we know the
next character we see could be the last and we need to be able to look back and see if k
characters before that we actually saw a zero. this requires a worst case exponential (in
k) size state diagram, in order to remember the k last characters.

3. Is the class of regular languages closed under intersection?

Yes. follow the same construction as for union, but now the final states of the new
automaton are the states (corresponding to subsets of states of the original automata)
that contain only accepting states from both automata (in the case of the union operation,
if any original state was accepting, then the new state is accepting)

4. Why is the class of regular languages closed under complement?

Yes, make sure the automaton deterministic (otherwise convert it to a deterministic one)
and then just switch the accepting and non-accepting states (accepting becomes non-
accepting and vice versa)

5. Given a non-deterministic automaton, describe how you would construct the automaton
that accepts the complement of the given NFA's language.

Convert it to a deterministic automaton and swap the accepting and non-accepting states

6. Converting NFAs to DFAs. Convert the following NFA to a deterministic one (ex1.41p57
from sipser)
7. Look through examples 1.56, 1.58 p68-69 (converting regular expressions to automata)

---

**Tutorial week 4**

Automata, regular languages. Some notes on minimizing automata and checking equivalence.

1. go through example 1.68p76 from sipser (converting a DFA to a regular expression)
2. Construct the NFA that accepts the same language as the following regular expression (using '+' for the union operation and nothing for concatenation):
   \[(a(a+b)^*)+(ba^*)\]
3. Find the equivalent regular expression for the following NFA

![Diagram of NFA]

4. Check whether the following two automata are equivalent, ie if they accept the same language

![Diagram of two automata]
5. Prove that the following examples are not regular (examples from the textbook, pages 80-82)
   - L1 = { all strings with n zeros followed by n ones, for all n >=0 }
   - L2 = { strings that have an equal number of ones and zeros }
   - L3 = { ww | w is any 0-1 string }
   - L4 = { lots of zeros followed by lots of ones, as long as we have more zeros than ones }

6. (advanced topics) Is the inverse of the pumping lemma for regular languages true? In other words is the pumping lemma an "if-and-only-if" statement?
   
   **Hint:** Here is a non-regular language that still works well with the pumping lemma:
   
   \[ L = \{ c^{n+k} a^n b^n \mid n > 0 \text{ and } k \geq 1 \} \]

7. (advanced topic) Is the following statement true or false? All grammars in Chomsky normal form are unambiguous.
   
   **Hint:** Ambiguous grammars exist. Inherently ambiguous grammars exist. All grammars can be converted to Chomsky normal form. Therefore there must be ambiguous grammars in Chomsky normal form.

---

**Tutorial week 5**

Topics: Turing machines

1. Discuss assignment 1 problem statements.
2. Give a detailed proof of the equivalence of two-stack automata and Turing machines. (week 4 lecture slides for definitions)
3. Is a non-deterministic 2-stack automaton more powerful than a deterministic one?
4. Prove that a 2 counter machine is equivalent to a Turing machine (week 4 lecture slides for definitions)
5. Go through the proofs of theorems 4.2-4.9 pages 167-172 from sipser
6. Construct a Turing machine that computes the proper subtraction operation ('monus' operation): \( a - b = \max(a-b, 0) \). Assume that the TM starts its computation with "$a$b" on its tape, where \( a \) and \( b \) are in unary, and you would like to finish the computation with "$c" on the tape where \( c = a - b \) (proper subtraction). Give a high-level description of the machine as well as a state diagram description and transition table
7. (Advanced topic) What would be an equivalent to 'undecidable problems' in lambda calculus?

---

**Tutorial week 6**

Topics: minimizing automata, state equivalence (these topics are from the week 4 tutorial questions)

Some [notes](#) on minimizing automata and checking equivalence.

1. Find the equivalent regular expression for the following NFA

   ![NFA Diagram]

2. Check whether the following two automata are equivalent, ie if they accept the same language
Hint: check if the two starting states are indistinguishable using the table filling method (as for minimizing automata)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

(X marks pairs of states that can be distinguished by an input string). A and C are indistinguishable therefore the automata accept the same language.

**Tutorial week 7**

1. Reductions and NP-completeness
2. the Clique problems is NP-complete (see lecture notes/slides)
3. The vertex cover problem is NP-complete (see lecture notes/slides)

**Tutorial week 8**

Logspace, non-deterministic logspace and logspace completeness.

**Tutorial week 9**
topics: complexity theory

NP-completeness proofs.

Discussing the assignment 1 answers, and possible questions clarifications for assignment 2.

Additional topics:

Between feasibility and hardness. Examples of problems that switch from easy to hard (P to NP-complete) for different values of parameters.

1. The bipartite matching problem is defined as follows: consider a bipartite graph with n vertices on each side (2n total). Is it possible to pick n edges such that every vertex is covered exactly once? A vertex is covered if it is an end-point of a chosen edge. Such an edge-set is called a matching. A more careful definition of the bipartite matching problem: Given disjoint sets X, and Y, each of size n, and given a set T, subset of X*Y (cartesian product of X,Y; a set of ordered pairs, that contain one element from each set) does there exist a set of pairs in T so that each element of X+Y (union of the sets) is contained in exactly one of these pairs?

Recall that the bipartite matching problem (or '2 dimensional' matching) can be solved in polynomial time (how?).

Prove that the 3 dimensional matching problem is NP-complete.

The 3D-Matching problem is defined as follows. Given disjoint sets X,Y and Z, each of size n, and given a set T subset of X*Y*Z (cartesian product of X,Y,Z; a set of ordered triples, that contain one element from each set) does there exist a set of triples in T so that each element of X+Y+Z (union of the sets) is contained in exactly one of these triples?

**Tutorial week 10**

topics: complexity theory and algorithms. Quiz will be during the first hour of lecture time. Solutions for the quiz questions will be discussed during the tutorial time.

**Tutorial week 11**

Topics: complexity, Between feasibility and hardness.
1. The **k-coloring** problem for a graph is defined as follows. Given an undirected graph $G=(V,E)$, and $k$ different colors, find a coloring of its vertices, such that adjacent vertices have different colors. $k$-coloring is in polynomial time for $k=2$. For $k=3$ and up, the problem becomes NP-complete. Prove that the **3-coloring** problem is NP complete.

2. 3SAT is NP-complete. 2SAT is the satisfiability problem where each clause has at most 2 literals. 2SAT is in P. Discuss how you can solve 2SAT in polynomial time

3. (adv) Something interesting for further reading (not part of the course topics): How many colors do you need to color a planar graph? Maybe 4 are enough for any planar graph? See the **4-color theorem** by Appel and Haken.

---

**Tutorial week 12**

topics: complexity and randomization. review. Exam structure will be discussed as well as what topics were covered.

*Tasos Viglas, University of Sydney. Last Update: Semester 2, 2007*