A. Regular expression and finite automata.

1. Give non-deterministic (with and without epsilon transitions), deterministic and minimal finite automata accepting the languages defined by each of the following regular expression:

   (a) \((ab + \epsilon)(aab + \epsilon)\)∗
   (b) \((a + b)^*aba(a + b)^*\)
   (c) \((a + b)^*(aba)^*(ab^*)^*\)

   The automata representations are given Figure 1(a-d) for (a), Figure 1(e-h) for (b) and Figure 2 for (c).

2. Give a regular expression representing the language over the alphabet \(\{a, b\}\) where if a (maximal consecutive) sequence of \(a\)'s of length \(m > 0\) is followed by a sequence of \(b\)'s of length \(n > 0\) then:
   - if \(m\) is odd then \(n\) is even and
   - if \(m\) is even then \(n\) is odd.

   Give equivalent epsilon-free and deterministic minimal automata.

   The language defined corresponds to the regular expression: \(b^*[((aa)^+b(bb)^* + a(aa)^*(bb)^+)^*a^*\). The automata representations are given Figure 3.

B. Weighted finite-state transducer and automata.

1. For each of the weighted automata and transducers over the tropical semiring given figure 4, specify whether they are trim, unambiguous and whether they have the twins property.
   - All the weighted automata and transducers are trim.
   - \(T_1, T_2, T_3\) and \(A_1\) are unambiguous. For each of these machines the unique final state have two incoming transitions \(e_1\) and \(e_2\) that don’t share the same input label \(i[e_1] = c \neq i[e_2] = d\). Moreover, the sub-machines corresponding to the set of successful ending in \(e_1\) (resp. \(e_2\)) are deterministic.
   - \(A_2\) is ambiguous since there are two successful paths with input \(abbc\).
   - \(A_3\) is ambiguous since there are two successful paths with input label \(abbbd\).

2. For which of these weighted automata does the theorems covered in class guaranty that:
   (a) the determinization algorithm terminates;
   (b) there exists no equivalent deterministic machine.

   Over the tropical semiring, having the twins property guarantees that termination of the determinization algorithm. Hence, \(T_1\) is determinizable. Moreover, if a machine is unambiguous, the three conditions:

   i) \(M\) has the twins property,
   ii) \(M\) is determinizable (determinization algorithm terminates) and...
Figure 1: (a) Thompson construction applied to \((ab + \epsilon)^*(aab + \epsilon)\), and result of (b) epsilon-removal, (c) determinization and (d) minimization. (e) Thompson construction applied to \((a + b)^*aba(a + b)^*\), and result of (f) epsilon-removal, (g) determinization and (h) minimization.
Figure 2: (a) Thompson construction applied to \((a + b)^*(aba)^*(ab^*)^*\), and result of (b) epsilon-removal, (c) determinization and (d) minimization.

Figure 3: (a) Thompson construction applied to \(b^*[(aa)^+(bb)^* + a(aa)^+(bb)^+]^*a^*\), and result of (b) epsilon-removal, (c) determinization and (d) minimization.
Figure 4: Weighted transducers (a) $T_1$, (b) $T_2$, (c) $T_3$ and weighted automata (d) $A_1$, (e) $A_2$ and (f) $A_3$ over the tropical semiring.

iii. $M$ admits an equivalent deterministic machine

are equivalent. Hence, $T_2$, $T_3$ and $A_1$ do not admit equivalent deterministic machine since they are unambiguous and don’t have the twins property.

3. (Extra credit) What can you say about the cases where the theorems do not apply. Would the determinization algorithm terminate? Does equivalent deterministic machine exist?

- The determinization algorithm would not terminate for $A_2$ as illustrated Figure ???. However there exists a deterministic automaton equivalent to $A_2$.
- .

C. Vowel restoration.

1. Install/compile the OpenFst library version 1.4.1 following the instruction at [http://openfst.org](http://openfst.org). A Linux environment is recommended; installation on Solaris or MacOS X is also possible. Note that there are Linux workstations you can access listed here (remote access) [http://cims.nyu.edu/webapps/content/systems/resources/computeservers] and here (at WWH) [http://cims.nyu.edu/webapps/content/systems/resources/labs]. Version 1.4.1 require a version of GCC greater than 4.6. Use version 1.3.4 if your version of GCC is too old. On most CIMS machines, the default version of GCC is 4.4.x but newer versions are also available. In that case, you can use 'CC=gcc49 CXX=g++49 ./configure' to use a more recent compiler.

2. Download the list of the 100 most common English words from: [http://en.wikipedia.org/wiki/Most_common_words_in_English](http://en.wikipedia.org/wiki/Most_common_words_in_English)

Let us denote $w_k \in \{a, b, \ldots, z\}^*$ the word of rank $k$ in that list and let $L = \{w_k|1 \leq k \leq 100\}$. 


3. We are going to use Zipf's law to obtain a (rough) estimate of the frequency of this words. Zipf's law states that the frequency of the word of rank $k$ is proportional to $1/k$. Give an expression of the relative frequency $f_k$ of $w_k$ as a function of $k$.

$$f_k = \frac{1}{k \sum_{i=1}^{100} \frac{1}{i}}$$

4. Use the OpenFst library to build the minimal weighted automata $A_1$ with alphabet $\Sigma = \{a, \ldots, z\}$ over the tropical semiring that accepts $w_k$ with weight $-\log f_k$ for all $1 \leq k \leq 100$. Restrict this construction to the 10 most frequent words and show the result.

A useful trick here is to generate a symbol table mapping each letter in \{a, b, \ldots, z\} to its ascii value but assigning the ascii value of space to the separator. To get nicer figures, 0 was assigned to the unicode value for $\epsilon$.

```bash
(echo -e "\0n_ 32";
 for ((i=97;$i<=122;i=$i+1)); do
   printf "\$(printf '%03o' $i); echo -e "$i";
 done) > syms

cat ranked_list |
gawk 'BEGIN {
  start = 0; ns = 1
  sum=0;
  for (i = 1; i<= 100; ++i)
    sum = sum + 1/i
}
{
  split($2, s,"");
  for (i = 1; i <= length(s); ++i) {
    if (i == 1) {
      p = 0; w = log($1) + log(sum)
    } else {
      p = ns - 1; w = 0
    }
    print p, ns, s[i], w
    ns = ns + 1
  }
  print ns - 1, 0.0
}' |
fstcompile --acceptor --isymbols=syms > a1.fst
```

5. From $A_1$, derive a weighted automaton $A_2$ with alphabet $\Sigma_\downarrow = \Sigma \cup \{\downarrow\}$ that accepts any (potentially empty) sequence of words in $L$ separated by $\downarrow$. More precisely, the weight associated to the sequence $w_{i_1} \downarrow w_{i_2} \downarrow \ldots \downarrow w_{i_l}$ should be $-\sum_{j=1}^{l} \log f_{i_j}$. Restrict this construction to the 10 most frequent words and show the result.
echo -e "0 1 \n0" | fstcompile --acceptor > space.fst
echo "0" | fstcompile > epsilon.fst
fstconcat space.fst a1.fst |
fstclosure |
fstconcat a1.fst - |
fstunion epsilon.fst - |
fstrmepsilon |
fstminimize |
fstacsort --sort_type=ilabel > a2.fst

6. Create a finite-state transducer \( T_1 \) with input and output alphabet \( \Sigma \cup \) that allows for arbitrary insertions of vowels in words in \( \Sigma^* \) (hint: this can be done with a single state).

```bash
cat syms |
gawk '
$2 > 0 {
    print 0, 0, $2, $2
    if (index("aeiou", $1) > 0)
        print 0, 0, 0, $2
}
END {
    print 0
}'} | fstcompile | fstarcsort --sort_type=olabel > t1.fst
```

7. Combine \( T_1 \) and \( A_2 \) to obtain a transducer \( T_2 \) that can be used to restore vowels that have been deleted from \( \sqcup \)-separated sequences of words in \( L \). Restrict this construction to the 10 most frequent words and show the result.

```bash
fstcompose t1.fst a2.fst | fstarcsort --sort_type=ilabel > t2.fst
```

8. Write 4 sentences using the words in \( L \). Delete most if not all vowels from these sentences. Show how the transducer \( T_2 \) can be used to restore the missing vowels. Try this approach on the sentences you generated and show the results.

```bash
echo " thnk cn mk ths wrk" |
farcompilestrings --far_type=fst --token_type="byte" --generate_keys=1 |
fstcompose - t2.fst |
fstproject --project_output |
fstshortestpath |
fstrmepsilon |
fstopsort |
fstprint |
gawk 'NF>2 {printf("%c",$3)} END {printf("\n")}'
```

9. What is your opinion on the performance of this approach? Do you have some suggestions on how to make this work better?
Figure 5: Weighted automata (a) $A_1$ and (b) $A_2$ restricted to the top 10 words. (c) Weighted transducer $T_1$. 