Lab 4: Finite State NLP

CS 474  Introduction to NLP
Mats Rooth

Introduction
This lab introduces finite state language processing using xfst. In the lab directory, there are files adj, vbase, and so forth which contain word lists for particular parts of speech. You will be reading these files into regular expressions, and building complex finite state machines (nets) based on these expressions.

Start the interface with the command xfst, and read a wordlist into a net, and print some random words:
```
read text vbase
print random-words
```

With print net > verb.net, create a text representation of the net in a file. Figure out the representation. How many states are there? How many final states?

Simple regular expressions
These commands define a simple finite state network, print the net, and visualize it.
```
read regex (a | b)* (c | d)* ;
print net
print vcg net
```

Figure out the functionality of apply down. Find the syntax for the following operations in the documentation, and construct a some examples: intersection, union, concatenation, set difference, iterative concatenation. How many words are there in the language (a | b)* (c | d)* which have length four or less?

Definitions
The following allows you define a name for a language.
```
read text vbase
define Vbase
```

The defined term can be used in regular expressions. Define a regex for the set of strings which are base form verbs in both directions, and list the words.
Using \textit{apply down}, verify that the expression

\[(b:\text{p} | ?)^*\]

optionally maps a b to a p. Using the term complement operator \textbackslash or another mehtod, define a relation which obligatorily maps a b to a p. Extend the solution so that each voiced oral stop consonant (b, d, or g) is obligatorily mapped to the corresponding unvoiced consonant (p, t, or k). Your regular relation should have the following behavior.

\begin{verbatim}
apply down> Your regular relation should have the following behavior.
Your regular relation should have the following behavior.
\end{verbatim}

Here is the idiom for applying a relation to a set, using compostion:

\begin{verbatim}
read regex Vbase .o. (b:p | ?)* ;
\end{verbatim}

Using this method, find the set of pairs of strings such that (i) the upper string is a base form verb containing a b, d, or k, (ii) the lower string is a base form verb, and (iii) the lower string is obtained from the upper string by devoicing b, d, and g. The solution should contain word pairs such as the following.

\begin{verbatim}
lo<b:p>
wa<g:k>e
sta<g:k>e
trea<d:t>
<d:t>rain
\end{verbatim}

\textit{Include the definition and some random words (or word pairs) in your handin.}

\textbf{Tokenization}

Suppose English were written without spaces:

\begin{verbatim}
parlousdosage
dividedhomily
prototypicaldistinction
\end{verbatim}

Define a relation which divides any adjective-noun combination such as this into two tokens by inserting a space when applied in the upward direction. It will be necessary to refer to the space character, which you can express using quotes, and the empty string, which is named with 0. Your relation should have the following behavior:

\begin{verbatim}
apply up> badidea
bad idea
apply up> relatedproposal
related proposal
apply up> proposalrelated
apply up>
\end{verbatim}

The substution operator -> could be used in the solution. \textit{In your handin, include your definition and some examples of multiple outputs for upward application.}

Exend the solution so that “bigbadwolf” is tokenized into three words.