1 Designing FSAs

(1) Vowel harmony (link) is a phonological pattern in which (simplifying greatly) all the vowels within some domain are identical.

Define a FSA that accepts a string if and only if all the vowels within a morpheme are identical. We will work with a very simple alphabet with one consonant `K`, two vowels `I` and `U`, and a symbol for morpheme boundaries `MB`. Thus, for example, `[I,K,I,K,MB,K,U,K,K,U]`, `[K,K,U,U,K]`, and `[MB,MB,K,MB,I,MB,U]` should all be accepted, but `[K,I,K,U]` should not be.

Check your answer using `parseFSA` (defined in `W8.hs`, and loaded by this file).

```
data SegmentKIU = K | I | U | MB deriving (Show,Eq)

fsaHarmony :: FSA SegmentKIU
fsaHarmony = (states, syms, i, f, delta)
  where
    states = undefined
    syms   = undefined
    i      = undefined
    f      = undefined
    delta  = undefined
```

(2) Define a FSA on the same alphabet accepting exactly the strings in which any `I`'s appear immediately after a `K`. This FSA should accept `[K,I]` and `[K,I,K,I]`, but not `[K,I,K,I,I]`, or `[K,I,MB,I]`.

Check your answer using `parseFSA`.

```
fsaKI :: FSA SegmentKIU
fsaKI = (states, syms, i, f, delta)
  where
    states = undefined
    syms   = undefined
    i      = undefined
    f      = undefined
    delta  = undefined
```

2 Parsing and forward sets

(3) A forward set for a string s, given a FSA m, is all of the ways that s can be parsed, according to m — in other words, the forward set contains the possible destination states one can arrive at by walking paths in m which (i) start in an initial state, and (ii) generate s.

Define a function `forward` that computes forward sets. It will be easiest to characterize your answer in terms of `walk`, which is defined in `W8.hs`, and loaded by this file.

```
forward :: Eq a => FSA a -> [a] -> [State]
forward (states, syms, i, f, delta) str = undefined
```

(4) Write a function `parseFSA2` which checks whether a FSA generates a string. Your answer should behave in exactly the same way as our existing parse function `parseFSA`, but be defined in terms of `forward`.

```
parseFSA2 :: Eq a => FSA a -> [a] -> Bool
parseFSA2 (q,syms,i,f,delta) str = undefined
```

(5) Recall the FSA from HW6 that accepts exactly the strings over `['a','b']` with an even number of `'a'`'s. Use `map` and `forward` to define `tested`, which computes the forward sets for the strings in `suite`.

```
evenas :: FSA Char
evenas = undefined

suite :: [String]
suite = ["aba", "aaa", "a", "b", "bb", "bbaabaa", "aaabbba", "ababa", "bbbbbaa"]

tested :: [[State]] -- a list of forward sets
tested = undefined
```

How many distinct forward sets do you observe? Does this make sense, given `evenas`? Why or why not?

```
{- Answer in one or two sentences here -}
```