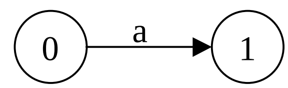
```
type FANode = (Int,       -- != (-1) = Final State
               [Int])     -- List of transitions.  Entries with destination of -1
                          -- are not defined (as in, they don't go anywhere)
newtype FA = F [FANode]  -- List of nodes with constructor
```

DFA’s are not very pretty to look at, here’s a simple example:

```
F [(-1,[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
1,-1,-1,-1,-1,-1]),
   (0,[-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
1,-1,-1,-1,-1,-1])]
```

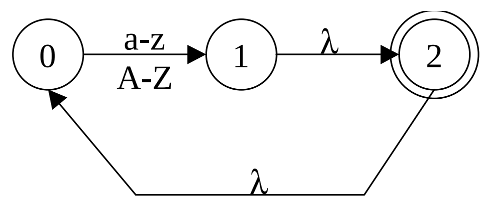
This is the representation of the DFA shown to the right. Alternatively, ‘`show`’n as:

```
Node #0: Non-Final - 1 transitions
        'a': to node #1
Node #1: Final[0] - 0 transitions
```



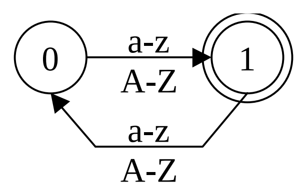
The first 256 entries of the transition list correspond to the ASCII values available for transitions. Values after these in the list correspond to lambda transitions. FA’s with lambda transitions are NFAs by definition. Here is an example:

```
Main> buildNFA "[a-zA-Z]+"
Node #0: Non-Final - 52 transitions
        'A' - 'Z' to node #1
        'a' - 'z' to node #1
Node #1: Non-Final - 1 transitions
        lambda to node #2
Node #2: Final[0] - 1 transitions
        lambda to node #0
```



This NFA may then be converted to a DFA [`compileRegEx = buildDFA . buildNFA`]:

```
Main> buildDFA $$
Node #0: Non-Final - 52 transitions
        'A' - 'Z' to node #1
        'a' - 'z' to node #1
Node #1: Final[0] - 52 transitions
        'A' - 'Z' to node #1
        'a' - 'z' to node #1
```



In addition to a list of transitions, each state contains a finality indicator. If this number is set to (-1), then the state is not final. If any other value, the state is final and that number is returned when a DFA is matched. This number is used by the lexer to determine which regular expression matched:

```
Main> compileLexer [("if", 0), ("[a-z]+", 1)] (-1)
Node #0: Non-Final - 26 transitions
        'a' - 'h' to node #1
        'i': to node #2
        'j' - 'z' to node #1
Node #1: Final[1] - 26 transitions        State matches “[a-z]+”
        'a' - 'z' to node #1
Node #2: Final[1] - 26 transitions        State matches “[a-z]+”
        'a' - 'e' to node #1
        'f': to node #3
        'g' - 'z' to node #1
Node #3: Final[0] - 26 transitions        State matches “if”
        'a' - 'z' to node #1
```

Lexer structure also contains a list that maps between RegEx ID’s and user ID’s:

```
newtype Lexer a = Lexr (FA,   -- Composite FA corresponding to all RE's
                        [a])  -- Mapping between DFA ID's and user defined
                              -- token ID's.  The error token is element 0.
```