1. `let x = 3 in let x = 7 in x * x`

7 * 7

49

2. `f ((\fn -> fn Paper) (\z -> whatItBeats z))`

f((\z → whatItBeats z) Paper)

f (whatItBeats Paper)

f Rock

1

3. `case (Win (whatItBeats Rock)) of {Draw -> m; Win z -> (m + f z)}`

case Win Scissors of {Draw → m; Win z → (m + f z)}

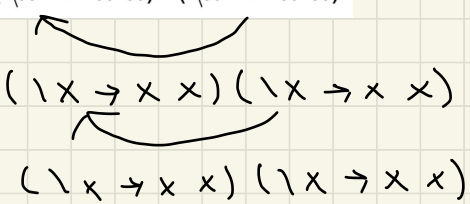
Win Scissors → (m + f Scissors)

4 + f Scissors

4 + 100

104

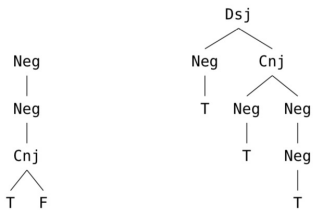
4. [optional] `(\x -> x x) (\x -> x x)`



We get an infinite loop.

2 Recursive functions on the `Form` type (6 points)

We can represent the structure of a `Form` with a tree, as illustrated by the following examples:



A. Write the expressions that these two trees represent.

1st Tree: Neg(Neg(Cnj T F))

2nd Tree: Dsj (Neg T) ((Cnj (Neg T)(Neg(Neg T))))

5b.

B. In this problem, you'll try designing some FSAs of your own. Provide a graphical representation for an FSA that has {C, V} as its alphabet and generates all and only those strings that contain at least two 'V's. Please submit a pdf for this. If you want to check whether your design is right, you can encode this FSA in Haskell as `fsa_twoVs :: Automaton SegmentCV`. It should behave like this:

```
*Assignment01> fsaSanityCheck fsa_twoVs
True
*Assignment01> generates fsa_twoVs [V,V,V,V]
True
*Assignment01> generates fsa_twoVs [V,C,V,C]
True
*Assignment01> generates fsa_twoVs [C,C,V,C]
False
*Assignment01> generates fsa_twoVs []
False
*Assignment01> generates fsa_twoVs [V]
False
```

two v's

two v's

one V

empty

one V