QUESTION 3

```
> type Grid = Matrix int
> type Matrix a = [[a]]
```

(a)

(i)

```
> choices :: Matrix int -> Matrix [int]
> choices xss = [[ if (x==0) then [1..9] else [x] | x <- xs] | xs <- xss]
```

(ii)

```
> choices' :: Matrix int -> Matrix [int]
> choices' = map (map (\x -> if (x==0) then [1..9] else [x]))
```

(b)

(i)

```
> cp :: [[a]] -> [[a]]
> cp [] = [[]]
> cp (xs:yss) = [ x:ys | x <- xs, ys <- cp yss]
```

(ii)

```
> cp' :: [[a]] -> [[a]]
> cp' = foldr f [[]]

> f :: [a] -> [[a]] -> [[a]]
> f xs yss = [x:ys | x <- xs , ys <- yss]
```

(c)

```
> expand :: Matrix [a] -> [Matrix a]
> expand = cp . map cp
```

(d)

```
> solve :: Grid -> [Grid]
> solve = (filter complete). expand . choices
```

The function "solve" will take into consideration 9^x grids (for an initial grid with x empty squares) as for each cell there are 9 possibilities that are formed in "choices" and each combination appears in "expand", later to be "filtered" with "complete".

(e)

```
> pruneRow :: [[Int]] -> [[Int]]
> pruneRow xss = map (remove (singleton xss)) xss

> singleton :: [[a]] -> [a]
> singleton xss = [x | [x] <- xss]

> remove :: [Int] -> [Int] -> [Int]
> remove xs [d] = [d]
> remove xs ds = filter (`notElem` xs) ds

> transpose :: [[a]] -> [[a]]
> transpose [] = []
> transpose [xs] = [[x] | x <- xs]
> transpose (xs:xss) = zipWith (:) xs (transpose xss)

> takeBy :: Int -> [a] -> [[a]]
> takeBy m [] = []
> takeBy m xs = take m xs : takeBy m (drop m xs)

> boxes :: [[a]] -> [[a]]
> boxes = map concat . concat . map transpose . takeBy 3 . map (takeBy 3)
```

Here, I did it with the help of Haskell because I repeatedly failed meeting the type signatures.

```
> prune :: [[[[Int]]]] -> [[[[Int]]]]
> prune = pruneBy boxes . pruneBy transpose . pruneBy id
>       where pruneBy h = h . map pruneRow . h
```

And the solver, which is not a very big improvement, but it is nevertheless faster:

```
> solveFast :: Grid -> [Grid]
> solveFast = filter complete . expand . prune . choices
```