Assignment 3 – Report

Name:	Chan(Chantel) Xu
Student Number:	u6233112
Course:	COMP1100
Date:	26/05/2017
Collaborators:	Jiafan Zhang (u6334938)
	Zixun Wu (u6193344)

Sudoku Project

- Sudoku.hs
  - The *allBlanks* function
  - The *isSudoku* function
  - The *noBlanks* function
  - The *printSudoku* function
  - The *fromString* function
  - The *toString* function
  - The *rows* function
  - The *cols* function
  - The *boxs* function
  - The *okBlock* function
  - The *okSudoku* function
  - The *blank* function
  - The *(!!=)* function
  - The *update* function
  - The *solve* function
- Testing
- Conclusion

- Reference

Sudoku.hs

Before writing functions, it was a problem for me to understand data types which represent Sudoku. After reading information about this part from Assignment 3 webpage, I knew that here defined a new data type (Sudoku) and the aim is to use the insert function. The Sudoku was a list of lists.

The *allBlanks* function

First, I used the worst way to write, which was like the *example* function. But when I saw the hints, I knew I could use *replicate* and then I searched for *replicate* in Hoogle. It was so simple that I could write in one line.

The *isSudoku* function

This function is designed to check whether the input Sudoku has the proper dimensions. I need to judge whether the input Sudoku has nine rows and whether each row has nine elements. I used *length* function to test whether the input Sudoku has nine rows and I wrote a new function to withdraw the elements and then use *length* to test it. The new function I wrote like this:

```
ele :: Sudoku -> Matrix Cell
ele (Sudoku s) = s
```

But by discussing with Jiafan Zhang, I could use *cells* function that given from line 36. Then I deleted *ele* function.

The *noBlanks* function

Firstly, I used *all* function which was advised in Assignment 3 webpage. But it always has error and I didn't know how to deal with it. I wrote it like this:

```
noBlanks :: Sudoku -> Bool
```

```
noBlanks (Sudoku s) = all (/= Nothing) s
```

Then I tried to write in another way using other functions: *notElem, concat.* They were easier to understand for me. The new one:

```
noBlanks :: Sudoku -> Bool
noBlanks (Sudoku s) = notElem Nothing (concat s)
```

The *printSudoku* function

The previous one:

```
printSudoku s = putStr (toString s)
```

The current one:

```
printSudoku s = putStr $ unlines $ chunksOf 9 (toString s)
```

I will explain the reason why I edited in *toString* part.

The *fromString* function

The previous one:

```
fromString str = Sudoku [map charToInt x | x <- lines str]
```

The current one:

```
fromString str = Sudoku (chunksOf 9 (map charToInt str))
```

I will explain the reason why I edited in *okSudoku* part.

The *toString* function

The previous one:

```
toString s = unlines [ map intToChar y | y <- cells s]
```

The current one:

```
toString s = map intToChar (concat(cells s))
```

After I had written the previous function, I run the command *doctest Sudoku*, the result was fault. That means one of my functions (*fromString, toString*) was wrong.

```
>doctest Sudoku
### Failed in Sudoku.hs:205: expression `fromString (toString s) = s`
```

After I rethought about this function, I found that I couldn't use *unlines*. Because it

joined lines with “\n”, which couldn’t make the result reach we expect. Moreover, I needed to edit the *printSudoku* function and *unlines* could be inserted in this function.

The *rows* function

It is easy.

The *cols* function

I used *transpose* function which was advised in Assignment 3 webpage. I searched for it in Hoogle and then I knew how to use it.

The *boxs* function

The previous one:

```
boxs :: Matrix a -> [Block a]
boxs [] = []
boxs m = box1 (take 3 m) ++ box1 (drop 3 m)
        where
           box1 [[],[],[]] = []
           box1 x = concat (map (take 3) x) : box1 (map (drop 3) x)
```

I tested this function like this:

```
> boxs (cells example)
[[Just 3,Just 6,Nothing,Nothing,Just 5,Nothing,Nothing,Nothing,Just 9],[Nothing,Just 7,Just 1,Nothing,Nothing,Nothing,Just 2,Nothing,Just 4],[Just 2,Nothing,Nothing,Just 1,Just 8,Nothing,Just 7,Nothing,Nothing],[Nothing,Nothing,Nothing,Just 4,Nothing,Nothing,Just 2,Just 7,Nothing,Nothing,Nothing,Just 5,Nothing,Just 8,Just 3,Nothing,Nothing,Just 7],[Nothing,Just 1,Just 3,Just 5,Nothing,Just 2,Just 4,Just 6,Nothing,Just 3,Nothing,Just 8,Nothing,Nothing,Nothing,Just 6,Just 9,Nothing],[Nothing,Just 2,Just 8,Nothing,Nothing,Just 9,Nothing,Nothing,Nothing,Just 9,Nothing,Nothing,Nothing,Just 6,Nothing,Nothing,Just 4,Just 3],[],[],[],[],[] ,[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],
```

```
[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]……
```

I didn't know why and I thought my thought was right. And then I went to study event to ask for help. After discussing with a mentor in study event, I knew that I just took many 3*3 boxes. The aim for *boxs* function was to take 3*3 elements and then let them be a list. Although I knew why I still didn't know how to write. I asked Zixun Wu for help and he told me that I could use list comprehension. The codes written by him were so concise to reach the function. I tried to write my own codes but they were complex. Finally, I referenced Zixun's codes. I understood them completely.

The current one:

```
boxs :: Matrix a -> [Block a]
boxs matrix = map concat [(map (take 3 . drop i).(take 3 . drop j)) matrix | i <- [0, 3, 6],
                    j <- [0, 3, 6]]
```

The *okBlock* function

I discussed three cases by empty list, the list with Nothing and the integer list.

The *okSudoku* function

I tested this function like this:

```
> okSudoku allBlanks
  True
>okSudoku$fromString
"36..712...5....18...92.47......13.284..5.2..927.46......53.89...83....6...769..43"
  False
>okSudoku$fromString
"364871295752936184819254736596713428431582679278469351645328917983147562127695843"
  False
```

These were not I expected. After I rethought about *okSudoku* and *fromString* functions, I found that *fromString* was wrong and I couldn't use *lines.* Because it broke a string up into a list of strings with "\n". Therefore, I should use another way to break up a string.

By discussing with Jiafan Zhang, I knew that I could use *chunksOf.* Then I searched for it in Hoogle and I imported *Data.List.Split* to use *chunksOf.*

The *blank* function

After I google list comprehension and understood it completely, I wrote this function using list comprehension. I wrote a helper function which input a list of lists and a position and output the element where the input position was. And then check whether this element was Nothing. In *index* function, I used *(!!)* which was a standard function in Haskell. It was so difficult that I have thought about it for almost two days.

The *(!!=)* function

I used *take, drop* and ++ to get the origin list and add the new value. Then we updated the list.

The *update* function

The previous one:

```
update :: Sudoku -> Pos -> Int -> Sudoku
update (Sudoku s) (y,x) i = Sudoku ((!!=) s (y,n))
   where
      n = (!!=) (s !! x) (x,Just i)
```

The current one:

```
update :: Sudoku -> Pos -> Int -> Sudoku
update (Sudoku s) (y,x) i = Sudoku ((!!=) s (y,n))
   where
      n = (!!=) (s !! y) (x,Just i)
```

After I have written all functions, I run the command *time cabal run examples/easy.txt* and the result was:

```
[1 of 2] Compiling Sudoku    (Sudoku.hs, dist/build/Sudoku/Sudoku-tmp/Sudoku.o)
Linking dist/build/Sudoku/Sudoku …
Running Sudoku …
```

```
real    0m3.749s
user    0m1.296s
sys     0m0.712s
```

I thought there must be a wrong function. Finally, I found there was a little fault which I marked above.

The *solve* function

For this part, I followed the instructions provided in Assignment 3 webpage. I discussed three cases and inserted *okSudoku, noBlanks, toString* and *update.* To make codes easy to read I wrote two helper functions. One converted the String input into a Sudoku. The other one updated origin Sudoku.

Testing

The *printSudoku* function

Input:

>>>printSudoku example

Expected Output:

3 6 . . 7 1 2 . .

. 5 . . . . 1 8 .

. . 9 2 . 4 7 . .

. . . . 1 3 . 2 8

4 . . 5 . 2 . . 9

2 7 . 4 6 . . . .

. . 5 3 . 8 9 . .

. 8 3 . . . . 6 .

. . 7 6 9 . . 4 3

Actual Output:

3 6 . . 7 1 2 . .

. 5 . . . . 1 8 .

. . 9 2 . 4 7 . .

. . . . 1 3 . 2 8

4 . . 5 . 2 . . 9

2 7 . 4 6 . . . .

. . 5 3 . 8 9 . .

. 8 3 . . . . 6 .

. . 7 6 9 . . 4 3

The *boxs* function

Input:

>>> boxs (cells example)

Expected Output:

[[Just 3,Just 6,Nothing,Nothing,Just 5,Nothing,Nothing,Nothing,Just 9],[Nothing,Nothing,Nothing,Just 4,Nothing,Nothing,Just 2,Just 7,Nothing],[Nothing,Nothing,Just 5,Nothing,Just 8,Just 3,Nothing,Nothing,Just 7],[Nothing,Just 7,Just 1,Nothing,Nothing,Nothing,Just 2,Nothing,Just 4],[Nothing,Just 1,Just 3,Just 5,Nothing,Just 2,Just 4,Just 6,Nothing],[Just 3,Nothing,Just 8,Nothing,Nothing,Nothing,Just 6,Just 9,Nothing],[Just 2,Nothing,Nothing,Just 1,Just 8,Nothing,Just 7,Nothing,Nothing],[Nothing,Just 2,Just 8,Nothing,Nothing,Just 9,Nothing,Nothing,Nothing],[Just 9,Nothing,Nothing,Nothing,Just 6,Nothing,Nothing,Just 4,Just 3]]

Actual Output:

[[Just 3,Just 6,Nothing,Nothing,Just 5,Nothing,Nothing,Nothing,Just 9],[Nothing,Nothing,Nothing,Just 4,Nothing,Nothing,Just 2,Just 7,Nothing],[Nothing,Nothing,Just 5,Nothing,Just 8,Just 3,Nothing,Nothing,Just 7],[Nothing,Just 7,Just 1,Nothing,Nothing,Nothing,Just 2,Nothing,Just 4],[Nothing,Just 1,Just 3,Just 5,Nothing,Just 2,Just 4,Just 6,Nothing],[Just 3,Nothing,Just 8,Nothing,Nothing,Nothing,Just 6,Just 9,Nothing],[Just

2,Nothing,Nothing,Just 1,Just 8,Nothing,Just 7,Nothing,Nothing],[Nothing,Just 2,Just 8,Nothing,Nothing,Just 9,Nothing,Nothing,Nothing],[Just 9,Nothing,Nothing,Nothing,Just 6,Nothing,Nothing,Just 4,Just 3]]

The *blank* function

Input:

>>> quickCheck prop_blank

Expected Output:

+++ OK, passed 100 tests.

Actual Output:

+++ OK, passed 100 tests.

The *isSudoku* function

Input:

>>>quickCheck prop_isSudoku

Expected Output

+++ OK, passed 100 tests.

Actual Output:

+++ OK, passed 100 tests.

Conclusion

I have just done this basic project by using backtracking method. For extension part, I tried to write some functions but they cannot shorten the time. I have no idea about that.

Reference

https://cs.anu.edu.au/courses/comp1100/assignments/03/

https://wiki.haskell.org/List_comprehension

https://www.haskell.org/hoogle/