COL 100M - Lab Exam 3 Solutions

```
open String
open Printf
open List
open Array

type cell =
| Value of int
| PossibleValues of int list

let size = 9;;
let minisize = 3;;

(* Helper function :
getList s [] returns a list with elements from 1 .. s *)
let rec getList s l =
        if (s <= 0) then l
        else (getList (s-1) l)@[s]


(* returns l with any instance of a removed. *)
let rec removeValue l a =
        match l with
        | [] -> []
        | hd::tl -> let z = removeValue tl a in
                        if hd = a then z
                        else hd::z

(* removes value as a potential value from all cells in the
given row of the sudoku puzzle.
Returns true if anything changes, false otherwise. *)
let eliminateValueRow sudoku value row =
        let flag = ref false in
        (for i = 0 to (size - 1) do
                let c = get (get sudoku row) i in
                        match c with
                        | PossibleValues l ->
                                let k = removeValue l value in
                                (set (get sudoku row) i (PossibleValues k);
                                flag := ((List.length l) != (List.length k)))
                        | _ -> ()
        done;
        !flag)

(* removes value as a potential value from all cells in the
given column of the sudoku puzzle.
Returns true if anything changes, false otherwise. *)
let eliminateValueCol sudoku value col =
```

```
        let flag = ref false in
        (for i = 0 to (size - 1) do
                let c = get (get sudoku i) col in
                        match c with
                        | PossibleValues l ->
                                let k = removeValue l value in
                                (set (get sudoku i) col (PossibleValues k);
                                flag := ((List.length l) != (List.length k)))
                        | _ -> ()
        done;
        !flag)

(* removes value as a potential value from all cells in the
given box of the sudoku puzzle.
Returns true if anything changes, false otherwise. *)
let eliminateValueBox sudoku value box =
        let row = (minisize * (box / minisize)) in
        let col = (minisize * (box mod minisize)) in
        let flag = ref false in
        (
        for i = row  to (row + minisize - 1) do
                for j = col to (col + minisize - 1) do
                        let c = get (get sudoku i) j in
                        match c with
                        | PossibleValues l ->
                                let k = removeValue l value in
                                (set (get sudoku i) j (PossibleValues k);
                                flag := ((List.length l) != (List.length k));
                                )
                        | _ -> ()
                done
        done;
        !flag)

(* If the cell [i,j] is a Value cell, then its value is eliminated from its row, column, and box.
Returns true if anything changes, false otherwise. *)
let eliminate sudoku i j =
        let z = get (get sudoku i) j in
        match z with
        | Value x ->        let f1 = (eliminateValueRow sudoku x i) in
                        let f2 = (eliminateValueCol sudoku x j) in
                        let f3 = (eliminateValueBox sudoku x ((i/minisize)*minisize + j/minisize)) in
                        f1 || f2 || f3
        | _ -> false

(* Runs a round of elimination for each cell in the puzzle.
Returns true if anything changes, false otherwise. *)
let eliminateAll sudoku =
        let flag = ref false in
        (for i = 0 to (size - 1) do
                for j = 0 to (size - 1) do
                        (* We need the let because || is lazily evaluated *)
                        let f1 = (eliminate sudoku i j) in
                        flag := (!flag) || f1
                done
        done;
```

```
        (!flag))

(* returns true if element a is in list l. May be replaed with a List.find. *)
let rec belongs l a =
        match l with
        | [] -> false
        | hd::tl -> if (hd = a) then true else belongs tl a

(* Returns a list of tuples of the form (row;col) which indicate the
cells in this row that have value as a candidate. *)
let getCellsRow sudoku row value =
        let cells = ref [] in
        (for i = 0 to (size - 1) do
                let z = get (get sudoku row) i in
                match z with
                | PossibleValues l -> if belongs l value then
                                        cells := (!cells)@[(row,i)] else ()
                | Value j -> ()
        done;
        !cells)

(* Returns a list of tuples of the form (row;col) which indicate the
cells in this column that have value as a candidate. *)
let getCellsCol sudoku col value =
        let cells = ref [] in
        (for i = 0 to (size - 1) do
                let z = get (get sudoku i) col in
                match z with
                | PossibleValues l -> if belongs l value then
                                        cells := (!cells)@[(i,col)] else ()
                | Value j -> ()
        done;
        !cells)


(* Returns a list of tuples of the form (row;col) which indicate the
cells in this box that have value as a candidate. *)
let getCellsBox sudoku box value =
        let cells = ref [] in
        let row = (minisize * (box / minisize)) in
        let col = (minisize * (box mod minisize)) in
        (for i = row  to (row + minisize - 1) do
                for j = col to (col + minisize - 1) do
                        let z = get (get sudoku i) j in
                        match z with
                        | PossibleValues l -> if belongs l value then
                                                cells := (!cells)@[(i,j)]
                                                else ()
                        | Value j -> ()
                done
        done;
        !cells)


let assign sudoku value row col =
        set (get sudoku row) col (Value value);
```

```
        eliminate sudoku row col


(* For each cell in the sudoku grid, checks if it has only one candidate left.
If so, assigns the candidate to that cell.
Returns true if anything changes, false otherwise. *)
let loneCells sudoku =
        let flag = ref false in
        (for i = 0 to (size - 1) do
                for j = 0 to (size - 1) do
                        let c = (get (get sudoku i) j) in
                        match c with
                        | Value j -> ()
                        | PossibleValues l -> if ((List.length l) = 1) then
                                                (* If the value assigned to the lone cell was also
                        a lone ranger, then assign returns false, even though
                        the grid has changed. So, force flag to true.*)
                                                (flag := assign sudoku (hd l) i j;  flag := true)
                                                else ()
                done
        done;
        (!flag))

(* Finds a lone ranger in row/ column/ box number id. f is a function, which is one of
getCellsRow / getCellsCol/ getCellsBox. If a lone ranger is found, it is assigned to its cell.
Returns true if anything changes, false otherwise. *)
let loneRanger sudoku id f =
        let flag = ref false in
        (for i = 1 to size do
                let cells = (f sudoku id i) in
                if (List.length cells) = 1 then
                (* In this case, assign will likely return false, even though the grid has changed.
                So, force flag to true. *)
                let c = (List.hd cells) in (flag := assign sudoku i (fst c) (snd c);
                                        flag := true)
                else ()
        done;
        !flag)

(* Runs loneRanger for each row, column, and box.
Returns true if anything changes, false otherwise. *)
let loneRangerAll sudoku =
        let flag = ref false in
        (for i = 0 to (size - 1) do
                (* Similar to eliminate, we use the lets because || is lazily evaluated,
        and also clarity. *)
                let f1 = (loneRanger sudoku i getCellsRow) in
                let f2 = (loneRanger sudoku i getCellsCol) in
                let f3 = (loneRanger sudoku i getCellsBox) in
                flag := (!flag) || f1 || f2 || f3
        done;
        (!flag))


let removePossibleValue sudoku row col value =
        let c = get (get sudoku row) col in
```

```
        match c with
        | Value x -> ()
        | PossibleValues l ->
                let k = removeValue l value in
                (set (get sudoku row) col (PossibleValues k))



(* Runs the pair heuristic for row/ column/ box number id.
f is a function, which is one of getCellsRow / getCellsCol/ getCellsBox.
If a pair is found, then all other candidates are removed from both cells.
Returns true if anything changes, false otherwise. *)
let getPair sudoku id f =
        let flag = ref false in
        let vals = getList size [] in
        let possibleCells = List.map (f sudoku id) vals in
        (for i = 0 to (size - 1) do
                for j = 0 to (size - 1) do
                        if (i != j)
                        && (List.length (nth possibleCells i) = 2)
                        && (List.length (nth possibleCells j) = 2) (* redundant, but stays for
            clarity *)
                        && (nth possibleCells i) = (nth possibleCells j) then
                                for k = 0 to 1 do (* this loop loops over both cells *)
                                        let cindex = (nth (nth possibleCells i) k) in
                                        let c = (get (get sudoku (fst cindex)) (snd cindex)) in
                                        match c with
                                        | PossibleValues l ->
                                                (* this loop removes all other candidates *)
                                                for x = 0 to ((List.length l)  - 1) do
                                                        let y = nth l x in
                                                        if (y != (i+1)) && (y != (j+1)) then
                                                                (removePossibleValue
                                sudoku (fst cindex) (snd cindex) y;
                                                                        flag := true)
                                                        else ()
                                                done
                                        | Value v -> ()
                                done
                done
        done;
        !flag)

(* Runs the pair heuristic for each row, column, and box.
Returns true if anything changes, false otherwise. *)
let getPairAll sudoku =
        let flag = ref false in
        (for i = 0 to (size - 1) do
                let f1 = (getPair sudoku i getCellsRow) in
                let f2 = (getPair sudoku i getCellsCol) in
                let f3 = (getPair sudoku i getCellsBox) in
                flag := (!flag) || f1 || f2 || f3
        done;
        (!flag))
```

```
(* Runs all heuristics to fill as many cells in the grid as possible,
and eliminate as many choices as possible. Does not return anything *)
let solveHumanistic sudoku =
        let flag = ref true in
        while ((!flag) = true && (not (isSolution sudoku))) do
                let flagE = eliminateAll sudoku in
                let flagLC = loneCells sudoku in
                let flagLR = loneRangerAll sudoku in
                let flagGP = getPairAll sudoku in
                flag := flagE || flagLC || flagLR || flagGP
        done

(* Returns (row,col) tuple -- the left most cell in the top most row that
is not a Value cell*)
let identifyTopCell sudoku =
        let row = ref (-1) in
        let col = ref (-1) in
        (for i = 0 to (size - 1) do
                for j = 0 to (size - 1) do
                        let c = get (get sudoku i) j in
                        match c with
                        | PossibleValues l -> if ((!row) = -1) && ((!col) = -1) then
                                                (row := i; col := j)
                                                else ()
                        | Value x -> ()
                done
        done;
        ((!row),(!col))
        )

(* Returns true if value can be filled in the cell (row,col),
i.e. it does not conflict with any of the other cells.
Used as a subroutine for brute force. *)
let bruteForceHelper sudoku value row col =
        let flag = ref true in
        (for i = 0 to (size - 1) do
                let c = get (get sudoku row) i in
                match c with
                | Value j -> if (j = value) then flag := false
                | PossibleValues l -> ()
        done;
        for i = 0 to (size - 1) do
                let c = get (get sudoku i) col in
                match c with
                | Value j -> if (j = value) then flag := false
                | PossibleValues l -> ()
        done;
        let r = minisize * (row/minisize) in
        let c1 = minisize * (col/minisize) in
        for i = r to (r + minisize - 1) do
                for j = c1 to (c1 + minisize - 1) do
                        let c = get (get sudoku i) j in
                        match c with
                        | Value j -> if (j = value) then flag := false
                        | PossibleValues l -> ()
```

```
                done
        done;
        (!flag))

(*
- Let c be the first unfilled cell in the sudoku.
- Fill c with one of its possible values, say x.
- Recursively solve sudoku using brute force.
- If no solution is arrived at with the value x in c,
        - fill the next possible value in c and repeat.
*)
let rec solveBruteForce sudoku =
        let k = identifyTopCell sudoku in
        if (fst k) = (-1) then true
        else
        let c = get (get sudoku (fst k)) (snd k) in
        match c with
        | PossibleValues l ->
                let flag = ref false in
                let i = ref 0 in
                (while ((!flag) = false) && ((!i) < (List.length l)) do
                        let x = (nth l (!i)) in
                        (if (bruteForceHelper sudoku x (fst k) (snd k)) then
                                (set (get sudoku (fst k)) (snd k) (Value x);
                                flag := solveBruteForce sudoku
                                );
                        i := (!i) + 1)
                done;
                if (!flag) = false then
                        (set (get sudoku (fst k)) (snd k) (PossibleValues l));
                (!flag)
                )
        | Value x -> true
```