```
A Simple Sudoku Solver
27th September, 2007
In Chapter 05
_________________________________________________________
0. Basic data types

> type Matrix a = [Row a]
> type Row a    = [a]

> type Grid     = Matrix Digit
> type Digit    = Char

> digits  :: [Digit]
> digits  =  ['1'..'9']

> blank   :: Digit -> Bool
> blank   =  (== '0')

1. Specification

> solve1 :: Grid -> [Grid]
> solve1 = filter valid . expand . choices

> type Choices = [Digit]

> choices :: Grid -> Matrix Choices
> choices = map (map choice)
>  where choice d | blank d   = digits
>                 | otherwise = [d]

> expand :: Matrix Choices -> [Grid]
> expand = cp . map cp

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

> valid  :: Grid -> Bool
> valid g = all nodups (rows g) &&
>           all nodups (cols g) &&
>           all nodups (boxs g)

> nodups       :: Eq a => [a] -> Bool
> nodups []     = True
> nodups (x:xs) = x `notElem` xs && nodups xs

> rows :: Matrix a -> [Row a]
> rows = id

> cols          :: Matrix a -> [Row a]
> cols [xs]     = [[x] | x <- xs]
> cols (xs:xss) = zipWith (:) xs (cols xss)

> boxs :: Matrix a -> [Row a]
> boxs = map ungroup . ungroup . map cols .
>        group . map group

> ungroup          = concat
> group []         = []
> group (x:y:z:xs) = [x,y,z]:group xs

2. Pruning

> prune :: Matrix Choices -> Matrix Choices
> prune =
>  pruneBy boxs . pruneBy cols . pruneBy rows
>  where pruneBy f = f . map pruneRow . f

> pruneRow :: Row Choices -> Row Choices
> pruneRow row = map (remove ones) row
>  where ones = [d | [d] <- row]

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

3. Single-cell expansion

> expand1   :: Matrix Choices -> [Matrix Choices]
> expand1 rows =
>  [rows1 ++ [row1 ++ [c]:row2] ++ rows2 | c <- cs]
>  where
>  (rows1,row:rows2) = break (any smallest) rows
>  (row1,cs:row2)    = break smallest row
>  smallest cs       = length cs == n
>  n                 = minimum (counts rows)

> counts = filter (/=1) . map length . concat

4. Final algorithm

> solve2 :: Grid -> [Grid]
> solve2 =  search . choices

> search :: Matrix Choices -> [Grid]
> search cm
>  |not (safe pm)  = []
>  |complete pm    = [map (map head) pm]
>  |otherwise      = (concat . map search . expand1) pm
>  where pm = prune cm

> complete :: Matrix Choices -> Bool
> complete = all (all single)

> single [_] = True
> single _   = False

> safe :: Matrix Choices -> Bool
> safe cm = all ok (rows cm) &&
>           all ok (cols cm) &&
>           all ok (boxs cm)

> ok row = nodups [d | [d] <- row]
```