Report: Assignment 3

Udbhav Chugh(170101081)

CS 431, Programming Language Lab

The steps to run and sample inputs for each of the three questions, Basic Set Operations, IITG Football League and House Planner are given in the README corresponding to each question. Refer it for information about code and steps to run.

1 Answers to questions asked for Problem 3: House Planner

1. **Write the algorithm (in pseudo-code) that you devised to solve the problem.**

Algorithm 1 House Planner Pseudo Code

function design($totalArea, bedroomCount, hallCount$)
    Intitialize all possible dimensions using Cartesian product for range of [$widthMin..widthMax$] X [$heightMin..HeightMax$]
    $bedroom$: [10..15] X [10..15]
    $hall$: [15..20] X [10..15]
    $kitchen$: [7..15] X [5..13]
    $bathroom$: [4..8] X [5..9]
    $balcony$[5..10] X [5..10]
    $garden$: [10..20] X [10..20]
    Initialise counts of each room (bedroom and hall count already given)
    $kitchenCount = ceil(bedroomCount/3)$
    $bathroomCount = bedroomCount + 1$
    $balconyCount = gardenCount = 1$
    Get all configurations of bedrooms and halls
    $possibleConfigs = bedroom$ X $hall, s.t., areasum <= totalArea$
    for $roomType$ in [$kitchen, bathroom, balcony, garden$] **do**
        if $roomType == Kitchen$ **then**
            $possibleConfigs = possibleConfigs$ X $kitchen$, s.t., $areasum <= totalArea$ &&
            $dim(Kitchens) <= dim(bedrooms)$ && $dim(Kitchens) <= dim(halls)$
        else if $roomType == Bathroom$ **then**
            $possibleConfigs = possibleConfigs$ X $bathroom$, s.t., $areasum <= totalArea$ &&
            $dim(bathrooms) < dim(kitchens)$
            Keep only one unique configuration per area in possibleConfigs to optimize time.
        else
            $possibleConfigs = possibleConfigs$ X $possibleDimensions$($balcony$ or $garden$), s.t., $areasum <= totalArea$
            Keep only one unique configuration per area in possibleConfigs to optimize time.
        end if
    end for
    $maxOccupiedArea = maximumAreaforpossibleConfigs$
    $finalConfigs = possibleConfigs, s.t, areaSum = maxOccupiedArea$
    $unusedSpace = totalArea - maxOccupiedArea$
    if $size(finalConfigs) > 0$ **then**
        $(bed, hall, kit, bath, bal, gar, area) = finalConfigs[0]$
        print the optimal configuration along with unsued space.
    else
        print (No design possible for the given constraints).
    end if
end function

2. **How many functions did you use?**
   I used a total of 10 functions. The function names and their purpose is listed below:

   - getInitialArea: get area occupied by a pair of bedroom and hall of a specific dimension
   - getArea: get updated area occupied when another room is added to the current configuration
   - checkArea: check if total area of the configuration is less than the required area
   - checkDimension: check if both dimensions of dim1 are less than dimensions of dim2. Used for maintaining size of kitchen smaller than hall and bedroom and size of bathroom smaller than size of kitchen
   - getMaxSpaceOccupied: among all possible configurations, get area which occupies maximum space
   - getoutputString: print final string in desired format
   - uniqueAreaBedHallKitBath: keep only one configuration of bed hall kitchen bathroom for each distinct area. This reduces search space as one configuration per area is enough to explore all possible configurations
   - uniqueAreaBedHallKitBathBal: keep only one configuration of bed hall kitchen bathroom and balcony for each distinct area.
   - uniqueAreaBedHallKitBathBalGar: keep only one configuration of bed hall kitchen bathroom balcony and garden for each distinct area
   - design: the design function which takes a tuple input and displays result on terminal. It computes all the Cartesian products and possible configurations as explained in pseudo-code. Example: design (1000,3,2)

3. **Are all those pure?**
   The design function is impure as it has IO () type as its return value. The presence of the IO type constructor means a function is impure. Any Haskell function that takes in input or displays output is impure as interaction with IO is involved. Hence the design function is impure.

   All other functions mentioned are pure functions. A pure function can be evaluated any number of times, with exactly same results each time without side-effects while an impure function might have side effects.

4. **If not, why? (Means, why the problem can't be solved with pure functions only)**
   In this problem, we need to display the final optimal configuration of rooms on the console for the user which is not possible without IO operation. And the presence of the IO type constructor means a function is impure. Hence this problem cannot be solved without the use of impure function.

2 Answers to additional questions

1. **Do you think the lazy evaluation feature of Haskell can be exploited for better performance in the solutions to the assignments? If so, which solution(s) and how?**

   Lazy evaluation is a method by which Haskell programs are evaluated. It means that evaluation of expressions is not done when they are bound to variables rather it is delayed until the results are needed. This avoids repeated and unnecessary evaluations. In general, it has various benefits like:

   - It avoids unnecessary expression evaluations which may not be used by the program.
   - It reduces overall (amortized) complexity since calculation is done only when needed.,
   - It allows programmer to focus on code and not worry about memory constraints since this lazy evaluation allows for extremely large data structures due to its postponing of evaluation nature.

   In regards to the questions in the assignments, there are a few instances where lazy evaluation feature can be exploited to give better performance. In question 2, at instance like searching for a particular team code in the the fixtures, lazy evaluation can ensure that only the fixtures

are traversed only till point that is required (lazy) instead of being eager which can increase unnecessary computations. Similarly in question3, we notice even though the question has very high order possibilities, using Haskell allows us to reduce the explored branches and evaluate only the necessary configurations and parts of code. In cases when we reach optimal value soon, or have multiple configurations evaluating to same area, lazy evaluation have reduce the branches to explore and avoid unnecessary computations.
Lazy propagation also allows us to not worry about memory constraints and generate large permutations while exploring branches as Haskell programs explores these permutations only when necessary. Thus it gives a lot of performance improvements.

2. **We can solve the problems using any imperative language as well. Do you find any advantage of using Haskell for these problems (w.r.t the property of lack of side effect)? If your answer is no, elaborate on why not?**
Yes, Haskell has various advantages over using any imperative language. In particular, most of the Haskell functions are pure, i.e., it can be evaluated any number of times, with exactly same results each time without "side-effect"(Except IO operations (which use an IO constructor to take input or display output to user), which are impure and can have side effects). Having pure functions and lack of side effects means that the functions will only modify values within the local environment and give same result on evaluation when executed any number of times. This offers several advantages:

   - This property of Haskell allows for easier code modifications. Since functions are pure and allow modifications in local environment, changing code has lower chance of cascading other errors into more functions. This makes it easier to refactor the code.
   - It offers easier testing and debugging solutions. Pure functions don't allow modifications for variables not in local environment. This allows the user to test these functions better individually without going through a series of functions.
   - Pure functions (lack of side effect property) allows user to focus on the algorithmic part of the program rather that handling implementation and minute error details which the Haskell systems handles well with pure functions.

   Apart from these benefits of lack of side effect property of pure functions, Haskell offers other advantages like lazy evaluation (discussed in previous question). Also Haskell has immutable variables which reduce possibility of error and allow for task parallelism. Also, Haskell programs have a mathematical function representation and allows user to translate pseudo code logic easily into code without worrying about minute implementation details. Hence Haskell has major advantages in terms of its property of lack of side effects and other advantages in general as well.