Based on the content of the sample exams and lecture notes, here is a customized exam focusing on declarative programming in Haskell and Prolog, with emphasis on functional and logic programming concepts.

University of Melbourne

Department of Computing and Information Systems
Declarative Programming
Sample Exam

Duration: 2 hours
Reading Time: 15 minutes
Total Marks: 100

Instructions:

1. Answer all questions on the exam paper itself.
2. The number of marks for each question is given; use it as a guide for the time to spend on each question.
3. No electronic devices or additional materials are permitted.

Question 1: Haskell Basics [16 marks]

For each of the following Haskell expressions, provide the type or indicate if it results in an error:

1.1. `:t (==)`

1.2. `:t map (*2)`

1.3. `map Just`

1.4. `:t (++ [1, 2])`

1.5. `\x -> x []`

1.6. `:t (++)`

1.7. `flip filter [1,2,3]`

1.8. `:t head`

Question 2: Haskell Evaluation [12 marks]

For each Haskell expression below, provide its value or explain why it results in an error. Assume `Data.List` is loaded.

2.1. `map length ["Hi", "World"]`

2.2. `filter (odd) [1,2,3,4,5]`

2.3. `let e = head [] in 5`

2.4. `map snd $ zip [1,2,3] ["a","b"]`

2.5. `foldr (:) [0] [1, 2, 3]`

2.6. `foldl (flip (:)) [] [1, 2, 3]`

Question 3: Functional Trees [20 marks]

Define a Haskell function `treeSum :: TTree Int -> Int` that calculates the sum of all values in a binary tree. Use the following type for binary trees:

```
data TTree a = Nil | Node a (TTree a) (TTree a)
```

Additionally, write a function `isBalanced :: TTree a -> Bool` that checks if a tree is balanced. A tree is balanced if, for each node, the height difference between its left and right subtrees is at most one.

Question 4: Prolog Basics [12 marks]

For each Prolog expression, state what will be printed or explain why it will fail:

4.1. `member(X, [1,2,3,4])`

4.2. `length([a,b,c], 4)`

4.3. `append([X|_], [Y|_], [1,2,3])`

4.4. `X = 3 * 7`

Question 5: Prolog List Operations [20 marks]

Write a Prolog predicate `reverse_list(L, R)` that reverses a list. For example:

```
?- reverse_list([1,2,3], R).
R = [3,2,1].
```

Additionally, write a Prolog predicate `is_palindrome(L)` that checks if a list is a palindrome. A list is a palindrome if it reads the same forwards and backwards.

Question 6: Functional Programming in Haskell [20 marks]

Write a Haskell function `subsequences :: [a] -> [[a]]` that returns all subsequences of a list in the order they appear. For instance:

```haskell
subsequences [1,2] = [[], [1], [2], [1,2]]
```

Additionally, implement `powerset :: [a] -> [[a]]` that returns the power set of a list, which includes all possible combinations of the list's elements.

Question 7: Prolog Tree Operations [20 marks]

Define a Prolog predicate `tree_member(X, Tree)` that checks if an element `X` exists in a binary tree. The tree will use the following structure:

```
tree(nil).
tree(node(Left, Value, Right)) :- tree(Left), tree(Right).
```

Also, write a predicate `tree_height(Tree, H)` that calculates the height of a binary tree, where `nil` has height 0, and a `node` has height 1 plus the maximum height of its subtrees.