COMP3141

Software System Design and Implementation

SAMPLE EXAM

Term 2, 2020

- Total Number of **Parts**: 5.
- Total Number of **Marks**: 125
- All parts **are** of equal value.
- Answer **all** questions.
- Excessively verbose answers may lose marks
- Failure to make the declaration or making a false declaration results in a 100% mark penalty.
- Ensure you are the person listed on the declaration.
- All questions must be attempted **individually** without assistance from anyone else.
- You must save your exam paper using the button below before time runs out.
- **Late submissions will not be accepted**.
- You may save multiple times before the deadline. Only your final submission will be considered.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Declaration

☐ I, name, declare that these answers are **entirely my own**, and that I did not complete this exam with assistance from anyone else.

Part A (25 Marks)

Answer the following questions in a couple of short sentences. No need to be verbose.

1. (3 Marks) What is the difference between a *partial function* and *partial application*?

   A partial function is a function not defined for its whole domain, whereas partial application is where a n-argument function is applied to fewer than n values.

2. (3 Marks) Name *two* methods of measuring program coverage of a test suite, and explain how they differ.

   Function coverage measures whether or not a given function is executed by the test suite, whereas path coverage measures all possible execution paths. Function coverage is easier to compute than path coverage.

3. (3 Marks) How are multi-argument functions typically modelled in Haskell?

Mutliple argument functions are usually modelled by one-argument functions that in turn return functions to accept further arguments. For example, the add function `(+) :: Int → (Int → Int)` takes an `Int` and returns a function.

4. (3 Marks) Is the type of *getChar* below a pure function? Why or why not?

$$getChar :: \texttt{IO Char}$$

It is not a pure function, but not because it is impure, but because it is not a function. It is instead an `IO` procedure.

Assignment Project Exam Help

5. (3 Marks) What is a *functional correctness* specification?

https://powcoder.com

A functional correctness specification is a set of properties that completely specify the definition of correctness for a program. It is often expressed as the combination of data invariants and a refinement from an abstract model.

Add WeChat powcoder

6. (3 Marks) Under what circumstances is performance important for an abstract model?

If we are testing using an abstract model (for example with QuickCheck), then we are still computing with it and thus intractable or uncomputable abstract models would not be suitable.

7. (3 Marks) What is the relevance of termination for the Curry-Howard correspondence?

The Curry-Howard correspondence assumes that function types are pure and total. Non-termination makes the logic the type system corresponds to inconsistent.

8. (4 Marks) Imagine you are working on some price tracking software for some company stocks. You have already got a list of stocks to track pre-defined.

$$\textbf{data}\ Stock = \text{GOOG} \mid \text{MSFT} \mid \text{APPL}$$
$$stocks = [\text{GOOG}, \text{MSFT}, \text{APPL}]$$

Your software is required to produce regular reports of the stock prices of these companies. Your co-worker proposes modelling reports simply as a list of prices:

$$\textbf{type}\ Report = [Price]$$

Where each price in the list is the stock price of the company in the corresponding position of the *stocks* list. How is this approach potentially unsafe? What would be a safer representation?

It is not guaranteed that the prices will line up to the stocks, or that there will be a stock for every price and a price for every stock. An alternative would be:

$$\textbf{type}\ Report = [(Stock, Price)]$$

which at least ensures that the right stocks are associated with the right price.

Assignment Project Exam Help

Part B (25 Marks)

https://powcoder.com

The following questions pertain to the given Haskell code:

Add WeChat powcoder

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f z (x : xs) = f x (foldr f z xs)  -- (1)
foldr f z []       = z                   -- (2)
```

1. (3 Marks) State the type, if one exists, of the expression $foldr\ (:)\ ([] :: [\text{Bool}])$.

   $[\text{Bool}] \to [\text{Bool}]$

2. (4 Marks) Show the evaluation of $foldr\ (:)\ []\ [1, 2]$ via equational reasoning.

$$\begin{aligned} & foldr\ (:)\ []\ [1,2] \\ = \quad & 1 : (foldr\ (:)\ []\ [2]) && (1) \\ = \quad & 1 : 2 : (foldr\ (:)\ []\ []) && (1) \\ = \quad & 1 : 2 : [] && (2) \end{aligned}$$

3. (2 Marks) In your own words, describe what the function $foldr\ (:)\ []$ does.

It is the identity function on lists.

4. (12 Marks) We shall prove by induction on lists that, for all lists xs and ys:

$$foldr\ (:)\ xs\ ys = ys\ ++\ xs$$

i. (3 Marks) First show this for the base case where $ys = []$ using equational reasoning. You may assume the left identity property for $++$, that is, for all ls:

$$ls = [] ++ ls$$

Assignment Project Exam Help
https://powcoder.com

$$\begin{aligned} foldr\ (:)\ xs\ [] &= xs && (2) \\ &= [] ++ xs && (id) \end{aligned}$$

Add WeChat powcoder

ii. (9 Marks) Next, we have the case where $ys = (k : ks)$ for some item k and list ks.

a. (3 Marks) What is the *inductive hypothesis* about ks?

$$foldr\ (:)\ xs\ ks = ks\ ++\ xs$$

b. (6 Marks) Using this inductive hypothesis, prove the above theorem for the inductive case using equational reasoning.

$$
\begin{array}{lcll}
foldr\ (:)\ xs\ (k : ks) & = & (:)\ k\ (foldr\ (:)\ xs\ ks) & (1) \\
& = & k : (foldr\ (:)\ xs\ ks) & \text{simp} \\
& = & k : (ks\ +\!\!+\ xs) & \text{I.H} \\
& = & (k : ks)\ +\!\!+\ xs & \text{def. of } (++)
\end{array}
$$

5. (2 Marks) What is the time complexity of the function call $foldr\ (:)\ []\ xs$, where xs is of size n?

   $\mathcal{O}(n)$

Assignment Project Exam Help

6. (2 Marks) What is the time complexity of the function call $foldr\ (\lambda a\ as \to as\ +\!\!+\ [a])\ []\ xs$, where xs is of size n

https://powcoder.com

   $\mathcal{O}(n^2)$

Add WeChat powcoder

Part C (25 Marks)

A *sparse vector* is a vector where a lot of the values in the vector are zero. We represent a sparse vector as a list of position-value pairs, as well as an `Int` to represent the overall length of the vector:

$$\mathbf{data}\ SVec = \mathsf{SV}\ \mathsf{Int}\ [(\mathsf{Int}, \mathsf{Float})]$$

We can convert a sparse vector back into a dense representation with this $expand$ function:

$$
\begin{array}{l}
expand :: SVec \to [\mathsf{Float}] \\
expand\ (\mathsf{SV}\ n\ vs) = map\ check\ [0..n-1] \\
\quad \mathbf{where} \\
\qquad check\ x = \mathbf{case}\ lookup\ x\ vs\ \mathbf{of} \\
\qquad\qquad \begin{array}{lcl}
\mathsf{Nothing} & \to & 0 \\
\mathsf{Just}\ v & \to & v
\end{array}
\end{array}
$$

For example, the $SVec$ value $\mathsf{SV}\ 5\ [(0, 2.1), (4, 10.2)]$ is expanded into $[2.1, 0, 0, 0, 10.2]$

1. (16 Marks) There are a number of $SVec$ values that do not correspond to a meaningful vector - they are invalid.
   i. (6 Marks) Which two *data invariants* must be maintained to ensure validity of an $SVec$ value? Describe the invariants in informal English.

   For a value SV n vs, the list vs should contain only one value for a given position, and the positions of all values in vs should all be ≥ 0 and $< n$.

   ii. (4 Marks) Give two examples of $SVec$ values which violate these invariants.

   Violates "only one value for a position": SV 2 $[(1, 1.1), (1, 5.3)]$
   Violates "all positions ≥ 0 and $< n$": SV 2 $[(3, 1.1)]$

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

   iii. (6 Marks) Define a Haskell function $wellformed :: SVec \to Bool$ which returns `True` iff the data invariants hold for the input $SVec value$. Your Haskell doesn't have to be syntactically perfect, so long as the intention is clear.
   You may find the function $nub :: (Eq\ a) \Rightarrow [a] \to [a]$ useful, which removes duplicates from a list.

$$
\begin{aligned}
&wellformed\ (\mathsf{SV}\ n\ vs) \\
&\quad = \mathbf{let}\ ps = map\ fst\ vs \\
&\quad\quad \mathbf{in}\ all\ (\lambda x \to x \geq 0\ \texttt{\&\&}\ x < n)\ ps \\
&\quad\quad \texttt{\&\&}\ nub\ ps == ps
\end{aligned}
$$

2. (9 Marks) Here is a function to multiply a $SVec$ vector by a scalar:

$$
\begin{aligned}
&vsm :: SVec \to Float \to SVec \\
&vsm\ (\mathsf{SV}\ n\ vs)\ s = \mathsf{SV}\ n\ (map\ (\lambda(p, v) \to (p, v * s))\ vs)
\end{aligned}
$$

i. (3 Marks) Define a function $vsmA$ that performs the same operation, but for dense vectors (i.e. lists of Float).

$$vsmA\ xs\ s = map\ (*\ s)\ xs$$

ii. (6 Marks) Write a set of properties to specify *functional correctness* of this function. *Hint:* All the other functions you need to define the properties have already been mentioned in this part. It should maintain data invariants as well as refinement from the abstract model.

Data invariants:

$$wellformed\ xs \implies wellformed\ (vsm\ xs\ s)$$

Data refinement:

$$expand\ (vsm\ xs\ s) == vsmA\ (expand\ xs)\ s$$

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Part D (25 Marks)

1. (10 Marks) Imagine you are working for a company that maintains this library for a database of personal records, about their customers, their staff, and their suppliers.

$$\textbf{newtype}\ Person = \ldots$$

$$name :: Person \to String$$
$$salary :: Person \to \text{Maybe}\ String$$
$$fire :: Person \to \text{IO}\ ()$$
$$company :: Person \to \text{Maybe}\ String$$

The *salary* function returns Nothing if given a person who is not a member of company staff. The *fire* function will also perform no-op unless the given person is a member of company staff. The *company* function will return Nothing unless the given person is a supplier.

Rewrite the above type signatures to enforce the distinction between the different types of person statically, within Haskell's type system. The function *name* must work with all kinds of people as input.
Hint: Attach *phantom* type parameters to the *Person* type.

```
data Staff
data Customer
data Supplier
newtype Person t = ...

name :: Person t → String
-- Maybe no longer needed here:
salary :: Person Staff → String
fire :: Person Staff → IO ()
-- Maybe no longer needed here:
company :: Person Supplier → String
```

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

2. (15 Marks) Consider the following two types in Haskell:

```
data List a where
  Nil   :: List a
  Cons  :: a → List a → List a

data Nat = Z | S Nat
data Vec (n :: Nat) a where
  VNil  :: Vec Z a
  VCons :: a → Vec n a → Vec (S n) a
```

What is the difference between these types? In which circumstances would *Vec* be the better choice, and in which *List*?

`Vec` tracks its length in the type level, whereas `List` does not. Usually, one would only use `Vec` if one frequently needed to express static pre-conditions about the length of the list. This way, the type checker can ensure these preconditions are met automatically.

i. (5 Marks)

ii. (5 Marks) Here is a simple list function:

$$
\begin{array}{lllll}
zip & :: & List\ a \to List\ b \to List\ (a, b) & & \\
zip & \mathsf{Nil} & ys & = & \mathsf{Nil} \\
zip & xs & \mathsf{Nil} & = & \mathsf{Nil} \\
zip & (\mathsf{Cons}\ x\ xs) & (\mathsf{Cons}\ y\ ys) & = & \mathsf{Cons}\ (x, y)\ (zip\ xs\ ys)
\end{array}
$$

Define a new version of zip which operates on Vec instead of $List$ wherever possible. You can constrain the lengths of the input.

Assignment Project Exam Help

$$
\begin{array}{lllll}
zip\,V & :: & Vec\ n\ a \to Vec\ n\ b \to Vec\ n\ (a, b) & & \\
zip\,V & \mathsf{VNil} & ys & = & \mathsf{VNil} \\
zip\,V & xs & \mathsf{VNil} & = & \mathsf{VNil} \\
zip\,V & (\mathsf{VCons}\ x\ xs) & (\mathsf{VCons}\ y\ ys) & = & \mathsf{VCons}\ (x, y)\ (zip\,V\ xs\ ys)
\end{array}
$$

https://powcoder.com

Add WeChat powcoder

iii. (5 Marks) Here is another list function:

$$
\begin{array}{l}
filter :: (a \to \mathsf{Bool}) \to List\ a \to List\ a \\
filter\ p\ \mathsf{Nil} = \mathsf{Nil} \\
filter\ p\ (\mathsf{Cons}\ x\ xs) \\
\qquad \mid p\ x \qquad\quad = \mathsf{Cons}\ x\ (filter\ p\ xs) \\
\qquad \mid otherwise \;=\; filter\ p\ xs
\end{array}
$$

Define a new version of $filter$ which operates on Vec instead of $List$ wherever possible.

We can't return a Vec because we don't know how long it will be.

$$
\begin{aligned}
&\mathit{filterV} :: (a \to \mathsf{Bool}) \to \mathit{Vec}\ n\ a \to \mathit{List}\ a \\
&\mathit{filterV}\ p\ \mathsf{VNil} = \mathsf{Nil} \\
&\mathit{filterV}\ p\ (\mathsf{VCons}\ x\ xs) \\
&\quad \mid p\ x \quad = \quad \mathsf{Cons}\ x\ (\mathit{filterV}\ p\ xs) \\
&\quad \mid \mathit{otherwise} \quad = \quad \mathit{filterV}\ p\ xs
\end{aligned}
$$

Part E (25 Marks)

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

1. (10 Marks) An applicative functor is called *commutative* iff the order in which actions are sequenced does not matter. In addition to the normal applicative laws, a *commutative* applicative functor satisfies:

$$f\ \langle\$\rangle\ u\ \langle*\rangle\ v = \mathit{flip}\ f\ \langle\$\rangle\ v\ \langle*\rangle\ u$$

   i. (2 Marks) Is the `Maybe` Applicative instance *commutative*? Explain your answer.

   Yes. If either u or v are Nothing, the result is Nothing on both sides of the law. If both are not Nothing, then the result is f applied to their contents.

   ii. (3 Marks) We have seen two different `Applicative` instances for lists. Which of these instances, if any, are *commutative*? Explain your answer.

   The ZipList instance obeys the law above - the size is the minimum of the two input lists, and each element is combined with f in the same order. The combinatorial instance does not. For example if $u = [2, 3]$ and $v = [4, 5]$ and $f = (*)$, then the left hand side of the law is $[8, 10, 12, 15]$ and the right hand side gives us $[8, 12, 10, 15]$ - they are different.

iii. (5 Marks) A *commutative* monad is the same as a commutative applicative, only specialised to monads. Express the commutativity laws above in terms of monads, using either **do** notation or the raw pure/bind functions.

$$
\begin{aligned}
&\textbf{do} \\
&\quad x \leftarrow u \\
&\quad y \leftarrow v \\
&\quad pure\ (f\ x\ y) \\
&= \\
&\textbf{do} \\
&\quad y \leftarrow v \\
&\quad x \leftarrow u \\
&\quad pure\ (f\ x\ y)
\end{aligned}
$$

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

2. (10 Marks) Translate the following logical formulae into types, and provide Haskell types that correspond to proofs of these formulae, if one exists. If not, explain why not.
   i. (2 Marks) $(A \vee B) \rightarrow (B \vee A)$

$$
\begin{aligned}
&f :: (Either\ A\ B) \rightarrow (Either\ B\ A) \\
&f\ (\mathsf{Left}\ x) = \mathsf{Right}\ x \\
&f\ (\mathsf{Right}\ x) = \mathsf{Left}\ x
\end{aligned}
$$

   ii. (2 Marks) $(A \vee A) \rightarrow A$

$$f :: (Either\ A\ A) \to A$$
$$f\ (\mathsf{Left}\ x) = x$$
$$f\ (\mathsf{Right}\ x) = x$$

iii. (3 Marks) $(A \wedge (B \vee C)) \to ((A \wedge B) \vee (A \wedge C))$

$$f :: (A, (Either\ B\ C)) \to (Either\ (A, B)\ (A, C))$$
$$f\ (a, \mathsf{Left}\ x) = \mathsf{Left}\ (a, x)$$
$$f\ (a, \mathsf{Right}\ x) = \mathsf{Right}\ (a, x)$$

Assignment Project Exam Help

iv. (3 Marks) $\neg((A \to \bot) \vee A)$

https://powcoder.com

$$f :: (\mathsf{Either}\ (A \to Void)\ A) \to Void$$

Add WeChat powcoder

This function cannot be implemented in Haskell or simply-typed lambda calculus as it states the law of the excluded middle is false. While constructive logics do not admit the law of the excluded middle, they do not refute it either.

3. (5 Marks) Here is a Haskell data type:

$$\begin{aligned} \mathbf{data}\ X \quad = \quad & \mathsf{First}\ ()\ \mathsf{A} \\ \mid \quad & \mathsf{Second}\ ()\ \mathsf{Void} \\ \mid \quad & \mathsf{Third}\ (\mathsf{Either}\ \mathsf{B}\ ()) \end{aligned}$$

Using known type isomorphisms, simplify this type as much as possible.

This is equivalent to (using plus for `Either` and times for products):

$$
\begin{aligned}
& (() \times A) + ((() \times \mathsf{Void}) + (B + ())) \\
= \quad & A + (\mathsf{Void} + (B + ())) \\
= \quad & A + (B + ()) \\
= \quad & (A + B + ()) \\
= \quad & \mathsf{Maybe}\ (\mathsf{Either}\ A\ B)
\end{aligned}
$$

END OF SAMPLE EXAM

(don't forget to save!)

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder