Started on Friday, 9 December 2022, 12:35 PM

State Finished

Completed on Friday, 9 December 2022, 12:55 PM

Time taken 20 mins 1 sec

Grade 9.75 out of 10.00 (97.5%)

Question 1

Partially correct

Mark 0.75 out of 1.00

Select TRUE statements about lambda calculus, Racket, Haskell, and Prolog.

Sel	ect	one	or	more	٠

✓	a.	Haskell has a	strong	static type	system. ✓
----------	----	---------------	--------	-------------	-----------

- b. Haskell encourages the use of total functions.
- c. Alpha-equivalent lambda terms can have different sets of free variables.
- d. Haskell employs a strict evaluation strategy.
- e. Racket has a strong static type system.
- f. In Racket, or and and are regular functions.
- g. In Prolog, a cut can only be used to optimise the search, but cannot affect the meaning of a predicate.
- h. Prolog has a strong static type system.
- 🔟 i. In Prolog, a predicate can not only check property of input, but also produce output in the form of variable substitutions. 🗸
- j. Racket employs a strict evaluation strategy.

Your answer is partially correct.

You have correctly selected 3.

The correct answers are:

Haskell has a strong static type system.,

Racket employs a strict evaluation strategy., Haskell encourages the use of total functions., In Prolog, a predicate can not only check property of input, but also produce output in the form of variable substitutions.

Question 2
Correct
Mark 2.00 out of 2.00

Consider there following program in Prolog:

```
animal(X) :- cat(X).
animal(boris).

cat(lion).
cat(X) :- hasTail(X), !, catchesMice(X).
cat(bob).

hasTail(snowball).
hasTail(fluffy).
hasTail(jack).

catchesMice(snowball).
catchesMice(fluffy).
catchesMice(spaniel).
```

Which of the following answers will be given to the query ?- animal(A)?

Select one or more:

- a. A=spaniel
- b. There will be no valid answers (false immediately).
- c. **true** (without any substitutions)
- d. A=bob
- ☑ e. A=snowball✔
- f. This query will loop indefinitely without producing any answer.
- g. A=jack
- i. A=fluffy
- ☑ j. A=boris❤

Your answer is correct.

The correct answers are:

A=lion,

A=snowball,

A=boris

Question 3

Correct

Mark 2.00 out of 2.00

Match each of the following Racket expressions with their corresponding value.

```
'(3 2 1 5 4)
(apply append (map reverse '((1 2 3) (4 5))))
(filter odd? (apply append '((1 2 3) (4 5))))
                                                            '(1 3 5)
(reverse (apply append '((1 2 3) (4 5))))
                                                            '(5 4 3 2 1)
(foldl * 1 '(1 2 3 4 5))
                                                            120
(length (map (lambda (l) (apply * l)) '((1 2 3) (4 5))))
                                                            2
(foldl (lambda (x z) (cons x '())) 0 '((1 2 3) (4 5)))
                                                            '((45))
(apply + (map length '((1 2 3) (4 5))))
                                                            5
(apply append (filter empty? '((1 2 3) (4 5))))
                                                            '()
(map (lambda (x) (apply * x)) '((1 2 3) (4 5)))
                                                            '(6 20)
(foldl (lambda (x z) x) 0 '((1 2 3) (4 5)))
                                                            (45)
```

Your answer is correct.

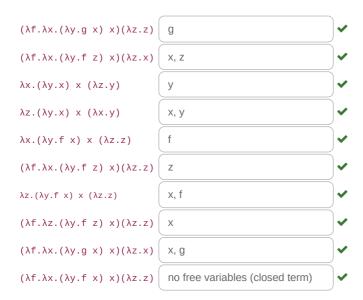
```
The correct answer is: (apply append (map reverse '((1 2 3) (4 5)))) \rightarrow '(3 2 1 5 4), (filter odd? (apply append '((1 2 3) (4 5)))) \rightarrow '(3 2 1 5 4), (foldl * 1 '(1 2 3 4 5)) \rightarrow 120, (length (map (lambda (l) (apply * l)) '((1 2 3) (4 5)))) \rightarrow 2, (foldl (lambda (x z) (cons x '())) 0 '((1 2 3) (4 5))) \rightarrow '((4 5)), (apply + (map length '((1 2 3) (4 5)))) \rightarrow 5, (apply append (filter empty? '((1 2 3) (4 5)))) \rightarrow '(), (map (lambda (x) (apply * x)) '((1 2 3) (4 5))) \rightarrow '(6 20), (foldl (lambda (x z) x) 0 '((1 2 3) (4 5))) \rightarrow '(4 5)
```

Question 4

Correct

Mark 2.00 out of 2.00

Which variables are free in each of the following lambda terms?



Your answer is correct.

The correct answer is: $(\lambda f.\lambda x.(\lambda y.g \ x) \ x)(\lambda z.z) \rightarrow g, (\lambda f.\lambda x.(\lambda y.f \ z) \ x)(\lambda z.x) \rightarrow X, Z, \lambda x.(\lambda y.x) \ x \ (\lambda z.y) \rightarrow y, \lambda z.(\lambda y.x) \ x \ (\lambda x.y) \rightarrow X, y, \lambda x.(\lambda y.f \ x) \ x \ (\lambda z.z) \rightarrow f, (\lambda f.\lambda x.(\lambda y.f \ z) \ x)(\lambda z.z) \rightarrow Z, \lambda z.(\lambda y.f \ x) \ x \ (\lambda z.z) \rightarrow X, f, (\lambda f.\lambda z.(\lambda y.f \ z) \ x)(\lambda z.z) \rightarrow X, (\lambda f.\lambda x.(\lambda y.g \ x) \ x)(\lambda z.x) \rightarrow X, g, (\lambda f.\lambda x.(\lambda y.f \ x) \ x)(\lambda z.z) \rightarrow no free variables (closed term)$

Question **5**

Correct

Mark 1.00 out of 1.00

Are you physically present in the room 108?

Select one:

■ True

False

The correct answer is 'True'.

```
Question 6
Correct
Mark 2.00 out of 2.00
```

Consider the following code in Haskell:

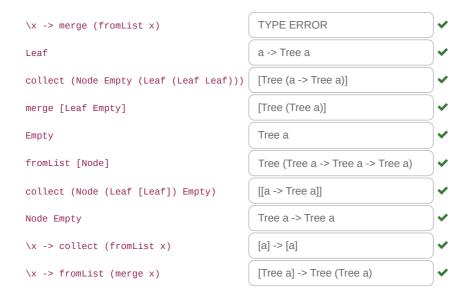
```
data Tree a = Empty | Leaf a | Node (Tree a) (Tree a)

collect :: Tree a -> [a]
collect Empty = []
collect (Leaf x) = [x]
collect (Node left right) = collect left ++ collect right

merge :: [Tree a] -> [Tree a]
merge (left:right:trees) = Node left right : trees
merge trees = trees

fromList :: [a] -> Tree a
fromList values = build (map Leaf values)
    where
    build [] = Empty
    build [tree] = tree
    build trees = build (merge trees)
```

Match the following expressions in Haskell with their corresponding types.



Your answer is correct.

The correct answer is: $\x -> \text{merge} \text{ (fromList } x) \to \text{TYPE ERROR, Leaf} \to a -> \text{Tree a, collect (Node Empty (Leaf (Leaf Leaf)))} \to [\text{Tree (a -> Tree a)}], \text{ merge [Leaf Empty]} \to [\text{Tree (Tree a)}], \text{ Empty} \to \text{Tree a, fromList [Node]} \to \text{Tree (Tree a -> Tree a, -> Tree a)}, \text{ collect (Node (Leaf [Leaf]) Empty)} \to [[a -> \text{Tree a}]], \text{ Node Empty} \to \text{Tree a, -> Tree a, -> collect (fromList x)} \to [a] -> [a], \times -> \text{ fromList (merge x)} \to [\text{Tree a}] -> \text{Tree (Tree a)}$