B00669954

HW3

CS571

Lovelin Anand Edward Paul

1.

a) Lexemes for compiler:

int

main

(

)

{

printf

(

“hello world\n “

)

;

return

0

;

}

b) Lexemes for syntax-directed editor:

/\* Introductory program \*/

\n

int

\s

main

(

)

\s

{

\n

\s

printf

(

“hello world\n“

)

;

\n

\s

return

\s

0

;

\n

}

2.

Though both operators prepend an element to an existing list, ‘:’ operator in Haskell can always prepend an element to a list while ‘cons’ operator in scheme always joins two elements to form a pair in which either of the element can be a list or other element.

: -> always list

cons -> always pair

3.

$ in Haskell is an infix operator which passes the expression in left as argument to expression in right.

In first expression,

map ($5) [\x->x + 3, \x->x - 3]

In second word ‘($5)’, the dollar sign will pass 5 as argument to any expression in the left. The map passes the lambda function from the list to this $ expression in turn, the dollar sign passes 5 as argument to each lambda function, which gets executed to give [8,2].

map ($5) [\x->x + 3, \x->x - 3]

* [(\x->x + 3)$5, (\x->x – 3)$5]
* [8,2]

But in second expression,

map (5$) [\x->x + 3, \x->x - 3]

5 is present before $ expression, the map will pass lambda function in list as argument to the $ expression, the $ expression will try to pass the lambda function as argument to 5 which raise error.

map (5$) [\x->x + 3, \x->x - 3]

* [(5$)(\x->x + 3), (5$)(\x->x – 3)]
* Error

4.

In Haskell,

Foldr

* foldr is right associative => f [x1,x2 …, xn] = f x1 ( f x2 ( … f xn) ) )
* Each argument as passed as argument to next function recursively.
* Since foldr is lazily evaluated and right associative, it is not needed to generate entire list before applying function to each member of the list.
* So foldr can be used to iterate infinite list.

Foldl

* Foldl is left associative => f[x1, x2, …, xn] = f (… (f (f x1) x2) …) xn
* It is tail recursive as it iterates the list completely before applying function.
* As foldl is right associative and tail recursive, The entire list has to be generated to apply function to its members.
* In infinite list, it is impossible to entire list . So foldl cannot be used to iterate infinite list.

5.

a)

**States** : Like Boolean datatype, An enumerated datatype can be created for state.

data State = Q0 | Q1 | Q2

**Transitions**: Each transition can be made as function expression. Since Haskell supports pattern matching, Displaying function in Haskell will be precise.

Transition :: State -> Int -> State

Transition Q1 1 = Q2

Transition Q1 0 = Q0

**DFA:** Similar to States, a new datatype can be created for DFA which accepts 3 –

Tuple

Data DFA = ( State, Transition, State)

b)

Transition Q0 0 = Q2

Transition Q0 1 = Q1

Transition Q1 0 = Q3

Transition Q1 1 = Q0

Transition Q2 0 = Q0

Transition Q2 1 = Q3

Transition Q3 0 = Q1

Transition Q3 1 = Q2

DFA = (Q0, Transition, Q0)

6.

a) f is injective total function:

Cardinality of A is less than or equal to cardinality of B

|A| <= |B|

b) f is surjective total function:

Cardinality of A is greater than or equal to cardinality of B

|A| >= |B|

c) f is bijective total function:

Cardinality of A is equal to Cardinality of B

|A| = |B|

7.

For given two finite sets A and B, there exists

|A|! or |B|! (since |A| = |B|) bijective total functions.

8.

So from below table,

a) p => p q

It is a tautology.

b) p => p q

It is **not** a tautology.

c) p q => p q

It is **not** a tautology.

d) p q => p q

It is a tautology.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **p** | **q** | **p q** | **p q** | **p => p q**  p (p q) | **p => p q**  p (p q) | **p q => p q**  (p q )( p q) | **p q => p q**  (p q )( p q) |
| True | True | True | True | True | True | True | True |
| True | False | True | False | True | False | False | True |
| False | True | True | False | True | True | False | True |
| False | False | False | False | True | True | True | True |

9.

1. **Invalid** – We can generate circular structure using lazy evaluation feature of Haskell.

cyclic = let x = 0 : y

y = 1 : x

in x

1. **Valid** – foldl can be implemented as tail recursive, as it is possible to iterate each element from the list using “:” operator
2. **Invalid** – Since it is impossible to find length of an infinite list , as it always keep running recursively, it is not possible to show there are more real numbers than natural numbers.
3. **Invalid** – Almost all datatypes can be compared for equality except builtin function datatype
4. **Valid –** In Set theory, functions are special kind of relations. So all functions are relation but inverse is not true.