CL exercise for Tutorial 2

Introduction

Objectives

In this tutorial you will:

- learn more about *types* and *predicates*;
- bring a small universe into Haskell and play with it;
- use quantifiers and compare natural language with Haskell.

Tasks

Exercises 1–5 are mandatory. Exercise 6 is optional, and exercise 7 is just for discussion.

Deadline

16:00 Tuesday 4 October

Reminder

Good Scholarly Practice

Please remember the good scholarly practice requirements of the University regarding work for credit.

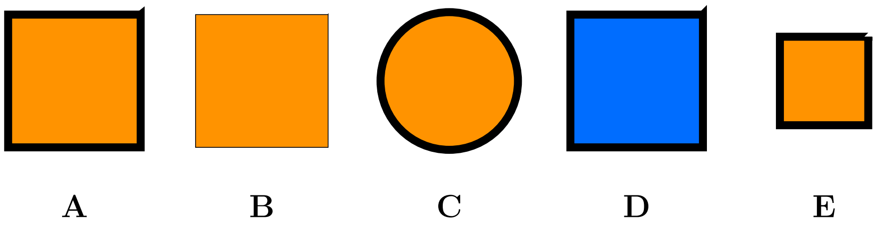
You can find guidance at the School page

https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct.

This also has links to the relevant University pages. Please do not publish solutions to these exercises on the internet or elsewhere, to avoid others copying your solutions.

Exercise 1 –mandatory—not marked–

Which of the things below is the odd one out? Why?



Solution to Exercise 1 *Finding the odd one out* in a set of things usually requires identifying some common features that all but one element in a set share.

Let's try to count the features that the 5 things have in common.

- Thing A has 3 features in common with each of the other things:
  - *colour*, *shape*, and *size* with B,
  - *colour*, *border*, and *size* with C,
  - *shape*, *size*, and *border* with D,
  - *colour*, *shape* and *border* with E.
- B, C, D, E on the other hand, share only 2 features with every other thing in our universe.

So it goes that the thing having the most features in common with the other things is the odd one out.

Exercise 2 –mandatory—marked–

Read Chapter 6 (*Features and Predicates*) from the textbook up to the end of *Checking Which Statements Hold*, page 49.

We will explore a little further ways of defining universes in Haskell.

Define a Haskell type of things and a list of all the things in the universe from Exercise 1, similarly to the way it is done in the textbook on page 46. Remember to include the magic `deriving (Eq,Show)`. Save your code in a file named `cl-tutorial-2.hs`

Identify the four features that characterize the things in Exercise 1.

Now create a type for each of the four features, as on p. 46, but also including `deriving (Eq,Show)`. Then write functions that, for each feature, give the feature of a thing.

(That is, if you have identified a feature 'fuzziness' with values 'soft' and 'hard', you should define a type `Fuzziness` with suitable values, and define a function `fuzziness :: Thing -> Fuzziness`.)

Solution to Exercise 2

We define the type of things and a list of all the things in the universe:

```haskell
data Thing = A | B | C | D | E deriving (Eq,Show)

things :: [Thing]

things = [ A, B, C, D, E ]
```

The four features that characterize the things in our universe are:

colour, shape, size, and border.

```
data Colour = Amber | Blue

colour :: Thing -> Colour

colour A = Amber
colour B = Amber
colour C = Amber
colour D = Blue
colour E = Amber

data Shape = Square | Circle

shape :: Thing -> Shape

shape A = Square
shape B = Square
shape C = Circle
shape D = Square
shape E = Square

data Size = Big | Small

size :: Thing -> Size

size A = Big
size B = Big
size C = Big
size D = Big
size E = Small

data Border = Thin | Thick

border :: Thing -> Border

border A = Thick
border B = Thin
border C = Thick
border D = Thick
border E = Thick
```

Exercise 3 –mandatory—not marked–

Because you defined your feature types `deriving (Eq,Show)`, you can compare feature values using equality `==` and inequality `/=`, which is very convenient.

First, add the `Predicate u` type to your file, as in lectures and the book.

Now define predicates, as in lectures and on p. 47. You can do this however you like, but it will be very easy if you use (in)equality of features.

(That is, for the hypothetical 'fuzziness' feature, you should define `isHard :: Predicate Thing` and `isSoft :: Predicate Thing`.)

```
isBlue :: Predicate Thing

isBlue x = colour x == Blue

isAmber :: Predicate Thing

isAmber x = colour x == Amber
```

and so on for the other features.

Exercise 4 –mandatory—marked–

We haven't yet covered existential statements ('Some x is y') in detail lectures. Read about the basics on page 48–49 of the book (which you should have done when starting this sheet . . .).

Express the following statements in Haskell using list comprehension and the five logical operations `&&`, `||`, `and`, `or`, `not` to combine the predicates defined above. Give the values of the Haskell expressions and check that they are correct according to the list of things in Exercise 1.

1. Every blue square has a thin border.
2. Some amber circle is not big.

Solution to Exercise 4

1. Every blue square has a thin border.

```
and [ hasThinBorder t | t <- things, isBlue t && isSquare t]
```

2. Some amber circle is not big.

```
or [ not (isBig t) | t <- things, isAmber t && isCircle t ]
```

Exercise 5 –mandatory—marked–

The statement "No square is blue" doesn't fit either of the patterns "Every X is Y" or "Some X is Y". But it is equivalent to a statement of the form "It is not the case that some X is Y" and also to a statement of the form "Every X is not Y".

Give those two equivalent statements and express them in Haskell using the five logical operations `&&`, `||`, `and`, `or`, `not`.

Solution to Exercise 5 The statement “No square is blue” is equivalent to:

- “It is not the case that some square is blue”:

  ```
  not (or [ isBlue t | t <- things, isSquare t ])
  ```

- “Every square is not blue”:

  ```
  and [ not (isBlue t) | t <- things, isSquare t ]
  ```

Exercise 6 –optional—marked–

Let’s stress again that optional exercises are *optional*. It gets you one mark, but is as much work as several of the previous. Do it only for fun, not for the mark!

We’ll now look at the answer (at least, our answer – did you agree?) to Exercise 1.

Write a function `thingsOtherThan` that returns, for every input `x`, the list of all remaining 4 things that are different from `x`.

```
thingsOtherThan :: Thing -> [Thing]
```

Complete the list of properties of things. It should have 8 elements.

```
properties :: [Predicate Thing]
properties = [isBlue, ...]
```

Solution to Exercise 6

```
thingsOtherThan :: Thing -> [Thing]
thingsOtherThan x = [ t | t <- things, t /= x ]



properties :: [Predicate Thing]
properties = [isBlue, isAmber, isSquare, isCircle, isBig,
              isSmall, hasThinBorder, hasThickBorder]
```

Next, write a function `propertiesOf` that returns, for every input `x`, the list of all properties of `x`. Note that every thing has exactly 4 properties.

```
propertiesOf :: Thing -> [Predicate Thing]
```

Write a function `isPropertyOfAnotherThing` that checks, for every predicate `p` and thing `x`, if `p` is a property of a thing different from `x`.

```
isPropertyOfAnotherThing :: Predicate Thing -> Thing -> Bool
```

Solution to Exercise 6 (cont.)

```
propertiesOf :: Thing -> [Predicate Thing]
propertiesOf x = [ prop | prop <- properties, prop x ]



isPropertyOfAnotherThing :: Predicate Thing -> Thing -> Bool
isPropertyOfAnotherThing prop x = or [ prop t | t <- thingsOtherThan x ]
```

Next, write a function `propertiesOnlyOf` that returns, for every input thing `x`, the list of all properties that are unique to `x`.

```
propertiesOnlyOf :: Thing -> [Predicate Thing]
```

Write a function `rank` that returns, for every input thing `x`, how many properties are unique to `x`. We call this number the *rank* of `x`.

```
rank :: Thing -> Int
```

Now look at the ranks of all the things in our universe. Which one stands out?

Solution to Exercise 6 (cont.)

```
propertiesOnlyOf :: Thing -> [Predicate Thing]

propertiesOnlyOf x = [ prop | prop <- propertiesOf x,
                             not (isPropertyOfAnotherThing prop x)]



rank :: Thing -> Int
rank x = length (propertiesOnlyOf x)
```

Exercise 7 –optional—not marked—for discussion–

The words “all”, “every”, “each”, “any”, and “some” are indicators of quantity. Generally they fall into two categories:

- indicators that refer to *all* things with a given property, or
- indicators that refer to *some* things with a given property.

"all", "every" and "each" belong to the first category, while "some" belongs to the second category. But "any" may be ambiguous.

Discuss with your colleagues the meaning of the word "any" in the following sentences.

1. Is there any amber square in our universe?
2. Any thing in our universe is amber.
3. We say that the universe is warm if any thing in it is amber.

Natural language is imprecise, but some languages are more precise than others. Can you think of similar examples of ambiguity related to "every", "any", and "some" from other languages?

Explain them to your group.

Use the universe in Exercise 1 to exemplify these ambiguities. Write the sentences in natural language, then disambiguate them by translation into Haskell code.