Part B: Non-interference

The label tracking system in Part B of this homework was particularly conservative in the labels it assigned to outputs of the hospital system’s components. The designers of the system only specified that at least one of the inputs to a component would be used in its output hence, without knowing which and without inspecting the code of these components, the tracking system was forced to assume that any or all of the inputs could have produced the output. Fortunately this simplicity meant that the labeling system could determine the proper output label solely from the input labels:

\[
\begin{array}{cccc}
\text{input: } l_1 & \text{input: } l_2 & \text{input: } l_3 & \text{...} \\
\text{Component} & & & \\
\text{output: } l_1 \sqcup l_2 \sqcup l_3 & & & \text{...}
\end{array}
\]

An overhaul of the hospital information system included fine-grained labels; instead of merely labeling the entire database containing patient information with the Profile label, a better interface provided access to data structures with components individually labeled according to their contents. Two such structures were those for representing bills and items within bills, below written in a Scala-like language with added label annotation. The code listing in this homework follow Scala syntax closely enough that readers can reference standard Scala syntax and semantics as an aide. The lattice for the labels used in this part of the homework is only the data lattice of Figure 1(a).

```scala
class Item {
    val quantity : int
    val priceEach : int
    val name : string
}

class Bill {
    val paid : boolean
    val items : List[Item]
    val patient : Patient
}
```

**UPDATED 2018-10-02:** Fixed typo in the text below previously stating that all of the fields of Item were labeled with Billing. Fixed to correctly state that only the quantity field is.

**UPDATED 2018-10-07:** Fixed typos in the code below referring to the field \texttt{costEach} which is actually called \texttt{priceEach}.

That is, an Item is an object with three fields: quantity of type int, priceEach of type int, and name of type string. The field quantity has label Billing, meaning it is considered Billing information. Likewise, a Bill has a boolean indicating whether the bill was paid, the list of items, and the patient to whom the bill belongs. Note that patient is labeled Profile, meaning it is considered Profile information. We will consider the label of a value to be part of its type. Any value without an explicit label is assumed to have label \bot.
Using such fine-grained labels, the billing department is able to query for the total outstanding bill amount, using the following map-reduce style method:

```python
def totalBill(bills: List[Bill]): int = {
    bills . filter { bill => ! bill . paid }
    . map { bill => bill . items . map { item => item . priceEach * item . quantity }
    . reduce { (a, b) => a + b } }
    . reduce { (a, b) => a + b }
}
```

This method employs the eponymous `map` and `reduce` methods as well as the related `filter`. Filter is a list method that returns a new list of only those items that satisfy the given function. Map produces a new list by applying a given function to each element. Reduce applies a binary method to pairs of elements repeatedly until there is only one left. Thus this method first filters out bills that have been paid, then maps each such bill to its total cost, and reduces to the sum of the costs.

Even though this method takes as input something labeled Profile, it does not use it. A sensible label tracking system would make use of this fact and not admit the possibility that Profile information is contained in the output of `totalBill`. To formalize this sensibility we rely on the definition of non-interference. First a preliminary definition:

**Definition 4** (View equality). Given values (tuples, lists, classes, etc.) $x, x'$ and label $L$, we say that they are $L$ equivalent or $x =_L x'$ when all of their components labeled with $L$ have the same value. Everything else can vary. We also assume that $x$ and $x'$ have the same structure and labels assigned to their constituent parts.

For example consider these items:

```plaintext
i1 = Item(quantity = 10,
    priceEach = 5,
    product = "Tylenol")
i2 = Item(quantity = 10,
    priceEach = 5,
    product = "Benadryl")
i3 = Item(quantity = 15,
    priceEach = 5,
    product = "Benadryl")
```

Then $i1 =_L i2$ for every label $L$ except ⊥. Likewise, $i1 =_L i3$ for every $L$ except Billing and ⊥. Recall the definition of `Item` presumes only the Billing label is placed upon the `quantity` field.

**Definition 5** (Interference). Given a method $m$ of type $X \rightarrow Y$ and a label $L$, we say that $L$ interferes with $m$ if there are two inputs of type $X$ that vary only in elements labeled $L$ or above (everything else is identical) that produce different outputs. That is, $\exists x : X, x' : X$ s.t. $x =_L x'$ for every $L' \not\supseteq L$ but $m(x) \not= m(x')$.

**Exercise 5.** On Line 3, the method `totalBill` uses a function to compute the cost of a given item based on its per unit price and its quantity. Let us give it a name and redefine it here:

```python
def itemTotal(item: Item): int = { item . priceEach * item . quantity }
```

(a) Given the definition of the Item class and the labels specified there, show that Billing interferes with `itemTotal`.

**Answer:** Items $i2$ and $i3$ above differ only in fields labeled Billing but produce the item totals of 50 and 75 respectively.

(b) Does Visits interfere with `itemTotal`? Why or why not?
Exercise 6. To aid in programming with labels, the developers of the tracking system decided to put labels on the outputs of methods to indicate which labels may interfere with them. That is: if a method \( m \) is given a type \( m: X \rightarrow Y_L \) then the label \( L \) or anything below \( L \) may interfere with the method, but nothing else does.

(a) Can the output of \texttt{totalBill} be given the label Profile?

\textbf{Answer:} Yes. Nothing not below Profile interferes with \texttt{totalBill}.

(b) What is the least label that can be given to the output of \texttt{totalBill}?

(c) The method \texttt{plus} is used repeatedly in the billing method.

\begin{verbatim}
def plus(a: int \_L_1, b: int \_L_2): int \_\_\_\_\_\_\_\_ = { a + b }
\end{verbatim}

In terms of the labels on its inputs, which label should (least) be assigned to the output of \texttt{plus}?

---

**UPDATED 2018-10-01:** Added the “Exercise 7” heading missing from the original posted version.

Exercise 7. The billing department also needs a means to query the outstanding balance for a particular patient. They create the method \texttt{totalBillForPatient} which first selects the bills for a given patient before summing up the balance:
def totalBillForPatient(patient: Patient, bills: List[Bill]): int
    totalBill =
        totalBill(
            bills.filter { bill => bill.patient == patient }
        )

(a) Which labels interfere with totalBillForPatient?

(b) What is the least label that can be assigned to its output?

(c) (BONUS) The types of the methods map, reduce, and filter, modified slightly to accept a list of elements
    of type T as their first argument, are as below.

```python
def map(list: List[T], fun: T => U): List[U] = ...
def reduce(list: List[T], fun: T => T => T): T = ...
def filter(list: List[T], fun: T => boolean): List[T] = ...
```

What are the least labels that can be assigned to the missing labels indicated by question marks:

```python
def map(list: List[T_L1], fun: T_L1 => U_L2): List[U_L3] = ...
def reduce(list: List[T_L1], fun: T_L1 => T_L2 => T_L3): T_L4 = ...
def filter(list: List[T_L1], fun: T_L1 => boolean_L2): List[T_L5] = ...
```

UPDATED 2018-10-07: Grammar fixed in the following exercise.