Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 4 з дисципліни

“Мультипарадигменне програмування”

Виконав студент: ІП-01 Танасієнко Олександр

Перевірив: ас. Очеретяний О. К.

Київ 2022

1. Завдання лабораторної роботи

Завдання :

1. Напишіть функцію `only_capitals` яка приймає на вхід `string list` та повертає `string list` що має тільки рядки що починаються з Великої літери. Вважайте, що всі рядки мають щонайменше один символ. Використайте `List.filter`, `Char.isUpper`, та `String.sub` щоб створити рішення в 1-2 рядки.

2. Напишіть функцію `longest_string1` що приймає `string list` та повертає найдовший `string` в списку. Якщо список пустий, поверніть `""`. У випадку наявності декількох однакових кандидатів, поверніть рядок, що найближче до початку списку. Використайте `foldl`, `String.size`, та ніякої рекурсії (окрім як використання `foldl` що є рекурсивним).

3. Напишіть функцію `longest_string2` яка точно така сама як `longest_string1` окрім як у випадку однакових кандидатів вона повертає найближчого до кінця кандидата. Ваше рішення має бути майже копією `longest_string1`. Так само використайте `foldl` та `String.size`.

4. Напишіть функції `longest_string_helper`, `longest_string3`, та `longest_string4` такі що:
- `longest_string3` має таку саму поведінку як `longest_string1` та `longest_string4` має таку саму поведінку як `longest_string2`.
- `longest_string_helper` має тип `(int * int -> bool) -> string list -> string` (зверніть увагу на curry). Ця функція буде схожа на `longest_string1` та `longest_string2` але вона є більш загальною так як приймає функцію як аргумент.
- Якщо `longest_string_helper` отримує на вхід функцію яка має поведінку як `>` (тобто повертає `true` тоді коли перший аргумент строго більше другого), тоді функція має таку саме поведінку як `longest_string1`.
- `longest_string3` та `longest_string4` є визначеними через `val`-прив'язки і часткове використання `longest_string_helper`.

5. Напишіть функцію `longest_capitalized` що приймає на вхід `string list` та повертає найдовший рядок в списку яка починається з Великої літери , або `""` якщо таких рядків немає. Вважайте, що всі рядки мають щонайменше один символ. Використовуйте `val`-прив'язки та ML бібліотечний `o` оператор для композиції функцій. Вирішіть проблему з однаковими результатами за прикладом завдання 2.

6. Напишіть функцію `rev_string`, що приймає на вхід `string` та повертає `string` що має ті самі символи в зворотньому порядку. Використайте ML `o` оператор, бібліотечну функцію `rev` для перевертання списків, та дві бібліотечні функції з `String` модулю. (Перегляньте документацію, щоб знайти найкращі підходящі)

Наступні дві проблеми передбачають написання функцій над списками які будуть використані в більш пізніх задачах.

7. Напишіть функцію `first_answer` типу `('a -> 'b option) -> 'a list -> 'b` (зауважте 2 аргументи curry). Перший аргумент має бути застосований до елементів другого аргументу до того моменту, як він поверне `SOME v` для деякого v і тоді v є результатом виклику `first_answer`. Якщо перший аргумент повертає NONE для всіх елементів списку, тоді має повернути виключення `NoAnswer`. Підказка: Приклад розв'язку має 5 рядків і не робить нічого складного.

8. Напишіть функцію `all_answers` типу `('a -> 'b list option) -> 'a list -> 'b list option` (зауважте 2 аргументи curry). Перший аргумент має бути застосований до елементів другого аргументу. Якщо результатом є NONE для будь якого з елементів, то результатом `all_answers` є NONE. Інакше виклики першого аргументу мають повернути `SOME lst1`, `SOME lst2`, ... `SOME lstn` та результатом `all_answers` буде `SOME lst` де `lst` є `lst1`, `lst2`, ..., `lstn` що складаються разом(порядок не важливий).

Підказки: Приклад розв'язку має 8 рядків. Він використовує допоміжні функції з акумулятором та `@`. Зауважте `all_answers f []` має отримати тип `SOME []`. Задачі що залишилися використовують наступні визначення типів, що були створені за образом вбудованої реалізації ML порівняння з шаблоном:

```
datatype pattern = Wildcard | Variable of string | UnitP |
ConstP of int | TupleP of pattern list | ConstructorP of
string * pattern
datatype valu = Const of int | Unit | Tuple of valu list |
Constructor of string * valu
```

Дано `valu` v та `pattern` p, або p співпадає з v або ні. Якщо так,співпадіння створює список `string * valu` пар; порядок в списку не має значення. Правила порівняння мають бути наступними:

- `Wildcard` співпадає з усім і створює пустий список прив'язок.
- `Variable s` співпадає з будь яким значенням `v` та створює одно елементний список що містить `(s,v)`.
- `UnitP` співпадає тільки з `Unit` та створює пустий список прив'язок.
- `ConstP 17` співпадає тільки з `Const 17` та створює пустий список прив'язок (так само для інших цілих чисел).
- `TupleP ps` співпадає з значенням форми `Tuple vs` якщо `ps` та `vs` мають однакову довжину і для всіх і, іий елемент `ps` співпадає з іим елементом `vs`. Список прив'язок що створюється в результаті є усіма списками вкладених порівнянь з шаблоном що об'єднані в один список.
- `ConstructorP(s1,p)` співпадає з `Constructor(s2,v)` якщо `s1` та `s2` є однаковою строкою (ви можете порівняти їх з =) та `p` співпадає з `v`. Список прив'язок створюється із вкладених порівнянь із шаблоном. Ми називаємо рядки `s1` та `s2` іменами конструкторів.
- Все інше не має значення.

9. (Ця задача використовує `pattern` тип даних але не зовсім про порівняння із шаблоном.) Функція `g` надана в **файлі**.
(1) Використайте `g` для визначення функції `count_wildcards`, що приймає на вхід pattern та повертає скільки `Wildcard` pattern-ів він містить.
(2) Використайте `g` для визначення функції `count_wild_and_variable_lengths` що приймає на вхід pattern та повертає кількість `Wildcard` `pattern`-ів які він містить плюс суму довжин рядків всіх змінних що містяться у змінній patterns. (Використайте `String.size`. Нам важливі тільки імена змінних; імена конструкторів не важливі.)
(3) Використайте `g` для визначення функції `count_some_var` що приймає на вхід строку та pattern (як пару) та повертає кількість входжень строки як змінної в pattern. Нам важливі тільки імена змінних; імена конструкторів не важливі.

10. Напишіть функцію `check_pat` що приймає на вхід pattern та повертає true тоді і тільки тоді коли всі змінні що з'являються в pattern відрізняються один від одного (наприклад, використовують різні рядки). Імена конструкторів не важливі. Підказки: Приклад розв'язку має 2 допоміжні функції. Перша приймає `pattern` та повертає список всіх рядків які він використовує для змінних. Використовуючи `foldl` з функцією яка використовує append може бути корисним. Друга функція приймає на вхід список рядків і вирішує чи він має повтори. `List.exists` може бути корисним. Приклад розв'язку має 15 рядків. Підказка: `foldl` та `List.exists` не обов'язкові, але можуть допомогти.

11. Напишіть функцію `first_match` що приймає на вхід value та список шаблонів та повертає `(string * valu) list option`, тобто `NONE` якщо ніякий паттерн зі списку не підходить або `SOME lst` де `lst` це список прив'язок для першого паттерну в списку який підійшов. Використайте `first_answer` та `handle`-вираз. Підказка: Приклад розв'язку має 3 рядки.

2. Програмний код

functions.sml

```
fun only_capitals(stringList: string list) =
    List.filter (fn str =>
                    Char.isUpper(String.sub(str, 0)))
                stringList;

fun longest_string1(stringList: string list) =
    List.foldl (fn (str1, str2) =>
                    if (String.size str1) > (String.size str2)
                    then str1
                    else str2)
                ""
                stringList;

fun longest_string2(stringList: string list) =
    List.foldl (fn (str1, str2) =>
```

```
                    if (String.size str1) >= (String.size str2)
                    then str1
                    else str2)
                ""
                stringList;

fun longest_string_helper stringComparator stringList =
    List.foldl (fn (str1, str2) =>
                    if stringComparator(String.size str1, String.size str2)
                    then str1
                    else str2)
                ""
                stringList;

val longest_string3 = longest_string_helper (fn (str1, str2) => str1 > str2);

val longest_string4 = longest_string_helper (fn (str1, str2) => str1 >= str2);

val longest_capitalized = longest_string1 o only_capitals;

val rev_string = String.implode o rev o String.explode;

exception NoAnswer;

fun first_answer functionToApply list =
    case list of
    head::tail => (case functionToApply(head) of
                    SOME v => v
                    |NONE => (first_answer functionToApply tail))
    |[] => raise NoAnswer;

fun all_answers functionToApply list =
    let
        fun helper(tl, accum) =
            case tl of
            head::tail => (case functionToApply(head) of
                            SOME v => helper(tail, accum @ v)
                            |NONE => NONE)
            |[] => SOME (accum)
    in
        helper(list, [])
    end;




datatype pattern = Wildcard
         | Variable of string
         | UnitP
         | ConstP of int
         | TupleP of pattern list
```

```
         | ConstructorP of string * pattern;

datatype valu = Const of int
          | Unit
          | Tuple of valu list
          | Constructor of string * valu;

fun g f1 f2 p =
    let
    val r = g f1 f2
    in
    case p of
        Wildcard          => f1 ()
      | Variable x        => f2 x
      | TupleP ps         => List.foldl (fn (p,i) => (r p) + i) 0 ps
      | ConstructorP(_,p) => r p
      | _                 => 0
    end;


fun count_wildcards(pat: pattern) =
    g (fn _ => 1)
      (fn _ => 0)
      pat;

fun count_wild_and_variable_lengths(pat: pattern) =
    g (fn _ => 1)
      (String.size)
      pat;

fun count_some_var(str: string, pat: pattern) =
    g (fn _ => 0)
      (fn strParam => if strParam = str
                      then 1
                      else 0)
      pat;

fun check_pat(pat: pattern) =
    let
        fun getStrVariables(patt: pattern) =
            case patt of
            Variable x => [x]
            |TupleP ps => List.foldl(fn (p1, acc) =>
                                        getStrVariables(p1) @ acc)
                                    []
                                    ps
            |ConstructorP(_,p) => getStrVariables(p)
            |_ => []
        fun checkDublicates(stringList: string list) =
            case stringList of
            [] => true
```

```
            |head::tail => if (List.exists (fn s => s = head) tail)
                           then false
                           else checkDublicates(tail)
    in
        checkDublicates(getStrVariables(pat))
    end;

fun first_match(someValue: valu, patternList: pattern list) =
    let
        fun matchHelper(v: valu, somePattern: pattern) =
            case (v, somePattern) of
            (_, Wildcard) => SOME []
            |(_, Variable s) => SOME [(s,v)]
            |(Unit, UnitP) => SOME []
            |(Const vLocal, ConstP p) => (if vLocal = p
                                          then SOME []
                                          else NONE)
            |(Tuple vs, TupleP ps) => (if (List.length vs) = (List.length ps)
                                       then all_answers matchHelper
(ListPair.zip(vs, ps))
                                       else NONE)
            |(Constructor (s1, vLocal), ConstructorP (s2, pLocal)) => (if s1 = s2
                                                                       then
matchHelper(vLocal, pLocal)
                                                                       else NONE)
            |(_, _) => NONE
    in
        SOME (first_answer (fn patParam => matchHelper(someValue, patParam))
patternList) handle NoAnswer => NONE
    end;
```

tests.sml

```
use "functions.sml";

(*only_capitals tests*)
fun provided_test_only_capitals() =
    let
        val strList1 = ["Fred", "Fredrick", "bicycle", "Elizabeth", "Betty",
"Freddie", "person"]
        val strList2 = ["London","England", "Poland"]
        val strList3 = ["car", "plane", "airport", "ship"]
    in
        [
            only_capitals(strList1),
            only_capitals(strList2),
            only_capitals(strList3)
        ]
    end;

provided_test_only_capitals();
```

```
(*longest_string1 tests*)
fun provided_test_longest_string1() =
    let
        val strList1 = []
        val strList2 = ["1111","2222", "3333"]
        val strList3 = ["car", "aircraft", "airport", "ship", "bicylcle"]
    in
        [
            longest_string1(strList1),
            longest_string1(strList2),
            longest_string1(strList3)
        ]
    end;

provided_test_longest_string1();

(*longest_string2 tests*)
fun provided_test_longest_string2() =
    let
        val strList1 = []
        val strList2 = ["1111","2222", "3333"]
        val strList3 = ["car", "aircraft", "airport", "ship", "bicylcle"]
    in
        [
            longest_string2(strList1),
            longest_string2(strList2),
            longest_string2(strList3)
        ]
    end;

provided_test_longest_string2();

(*longest_string3 tests*)
fun provided_test_longest_string3() =
    let
        val strList1 = []
        val strList2 = ["1111","2222", "3333"]
        val strList3 = ["car", "aircraft", "airport", "ship", "bicylcle"]
    in
        [
            longest_string3(strList1),
            longest_string3(strList2),
            longest_string3(strList3)
        ]
    end;

provided_test_longest_string3();

(*longest_string4 tests*)
fun provided_test_longest_string4() =
    let
```

```
        val strList1 = []
        val strList2 = ["1111","2222", "3333"]
        val strList3 = ["car", "aircraft", "airport", "ship", "bicylcle"]
    in
        [
            longest_string4(strList1),
            longest_string4(strList2),
            longest_string4(strList3)
        ]
    end;

provided_test_longest_string4();

(*longest_capitalized tests*)
fun provided_test_longest_capitalized() =
    let
        val strList1 = ["veryLongWord", "Fred", "Frederick", "bicycle",
"Elizabeth", "Betty", "Freddie", "person"]
        val strList2 = ["London","England", "Poland"]
        val strList3 = ["car", "plane", "airport", "ship"]
    in
        [
            longest_capitalized(strList1),
            longest_capitalized(strList2),
            longest_capitalized(strList3)
        ]
    end;

provided_test_longest_capitalized();

(*longest_capitalized tests*)
fun provided_test_rev_string() =
    let
        val str1 = "Elizabeth"
        val str2 = "London"
        val str3 = "airport"
    in
        [
            rev_string(str1),
            rev_string(str2),
            rev_string(str3)
        ]
    end;

provided_test_rev_string();

(*first_answer tests*)
fun provided_test_first_answer() =
    let
        val list1 = [5, 4, 3, 2, 1]
        val list2 = [4]
```

```
        val list3 = [1, 2, 3]
    in
        [
            (if (first_answer (fn arg => if arg = 4 then SOME (arg) else NONE)
list1) = 4 then "Right" else "NOT") handle (NoAnswer) => "Wrong",
            (if (first_answer (fn arg => if arg = 4 then SOME (arg) else NONE)
list2) = 4 then "Right" else "NOT") handle (NoAnswer) => "Wrong",
            (if (first_answer (fn arg => if arg = 4 then SOME (arg) else NONE)
list3) = 4 then "Right" else "NOT") handle (NoAnswer) => "Wrong"
        ]
    end;

provided_test_first_answer();

(*all_answers tests*)
fun provided_test_all_answers() =
    let
        val list1 = [5, 4, 3, 2, 1]
        val list2 = [4]
        val list3 = [1, 2, 3]
    in
        [
            all_answers (fn arg => if arg > 0 then SOME [arg] else NONE) list1,
            all_answers (fn arg => if arg = 4 then SOME [arg] else NONE) list2,
            all_answers (fn arg => if arg < 3 then SOME [arg] else NONE) list3
        ]
    end;

provided_test_all_answers();

(*count_wildcards tests*)
fun provided_test_count_wildcards() =
    let
        val pattern1: pattern = (TupleP ([ConstP(1),ConstP(1),
ConstructorP("mystr", Wildcard)]))
        val pattern2: pattern = Wildcard
        val pattern3: pattern = (TupleP ([Wildcard,ConstP(1), Wildcard,
Wildcard]))
        val pattern4: pattern = Variable "hello"
    in
        [
            count_wildcards(pattern1),
            count_wildcards(pattern2),
            count_wildcards(pattern3),
            count_wildcards(pattern4)
        ]
    end;

provided_test_count_wildcards();

(*count_wild_and_variable_lengths tests*)
```

```
fun provided_test_count_wild_and_variable_lengths() =
    let
        val pattern1: pattern = (TupleP ([Variable "car", Variable "bicycle",
ConstructorP("mystr", Wildcard)]))
        val pattern2: pattern = Wildcard
        val pattern3: pattern = (TupleP ([Wildcard, Variable "sometext",
Wildcard, Wildcard]))
        val pattern4: pattern = Variable "hello"
    in
        [
            count_wild_and_variable_lengths(pattern1),
            count_wild_and_variable_lengths(pattern2),
            count_wild_and_variable_lengths(pattern3),
            count_wild_and_variable_lengths(pattern4)
        ]
    end;

provided_test_count_wild_and_variable_lengths();

(*count_some_var tests*)
fun provided_test_count_some_var() =
    let
        val str1 = "car" val pattern1: pattern = (TupleP ([Variable "car",
Variable "car", ConstructorP("mystr", Wildcard)]))
        val str2 = "car"  val pattern2: pattern = Wildcard
        val str3 = "sometext"  val pattern3: pattern = (TupleP ([Wildcard,
Variable "sometext", Wildcard, Wildcard]))
        val str4 = "hello"  val pattern4: pattern = Variable "hello"
    in
        [
            count_some_var(str1, pattern1),
            count_some_var(str2, pattern2),
            count_some_var(str3, pattern3),
            count_some_var(str4, pattern4)
        ]
    end;

provided_test_count_some_var();

(*check_pat tests*)
fun provided_test_check_pat() =
    let
        val pattern1: pattern = (TupleP ([Variable "car", Variable "car",
ConstructorP("mystr", Wildcard)]))
        val pattern2: pattern = Wildcard
        val pattern3: pattern = (TupleP ([Wildcard, Variable "sometext",
Wildcard, Wildcard, Variable "another text"]))
        val pattern4: pattern = Variable "hello"
    in
        [
            check_pat(pattern1),
```

```
            check_pat(pattern2),
            check_pat(pattern3),
            check_pat(pattern4)
        ]
    end;

provided_test_check_pat();

(*first_match tests*)
fun provided_test_first_match() =
    let
        val valu1: valu = Const 5
        val pattern1: pattern list = [Variable "car", ConstP 5,
ConstructorP("mystr", Wildcard)]

        val valu2: valu = Unit
        val pattern2: pattern list = [UnitP]

        val valu3: valu = Constructor("str1", Const 1)
        val pattern3: pattern list = [Wildcard, Variable "sometext",
ConstructorP("str1", ConstP 1)]

        val valu4: valu = Tuple ([Const 5, Const 1])
        val pattern4: pattern list = [TupleP ([ConstP 5, Variable "car"])]
    in
        [
            first_match(valu1, pattern1),
            first_match(valu2, pattern2),
            first_match(valu3, pattern3),
            first_match(valu4, pattern4)
        ]
    end;

provided_test_first_match();
```

3. Результати виконання тестів

```
val provided_test_only_capitals = fn : unit -> string list list

val it =
  [["Fred","Fredrick","Elizabeth","Betty","Freddie"],
   ["London","England","Poland"],[]] : string list list

val provided_test_longest_string1 = fn : unit -> string list

val it = ["","1111","aircraft"] : string list

val provided_test_longest_string2 = fn : unit -> string list

val it = ["","3333","bicylcle"] : string list

val provided_test_longest_string3 = fn : unit -> string list

val it = ["","1111","aircraft"] : string list

val provided_test_longest_string4 = fn : unit -> string list

val it = ["","3333","bicylcle"] : string list

val provided_test_longest_capitalized = fn : unit -> string list

val it = ["Frederick","England",""] : string list

val provided_test_rev_string = fn : unit -> string list

val it = ["htebazilE","nodnoL","tropria"] : string list

val provided_test_first_answer = fn : unit -> string list

val it = ["Right","Right","Wrong"] : string list
```

```
val provided_test_all_answers = fn : unit -> int list option list

val it = [SOME [5,4,3,2,1],SOME [4],NONE] : int list option list

val provided_test_count_wildcards = fn : unit -> int list

val it = [1,1,3,0] : int list

val provided_test_count_wild_and_variable_lengths = fn : unit -> int list

val it = [11,1,11,5] : int list

val provided_test_count_some_var = fn : unit -> int list

val it = [2,0,1,1] : int list

val provided_test_check_pat = fn : unit -> bool list

val it = [false,true,true,true] : bool list

val provided_test_first_match = fn : unit -> (string * valu) list option list

val it = [SOME [("car",Const 5)],SOME [],SOME [],SOME [("car",Const 1)]] :
  (string * valu) list option list
-
```