Міністерство освіти і науки України
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

ЗВІТ

до лабораторних робіт №4

Мультипарадигменне програмування

Виконав студент ІПз01 Максиміхін О.В.
(№ групи, прізвище, ім’я, по батькові)

Прийняв ас. Очеретяний О. К.
(посада, прізвище, ім’я, по батькові)

Київ 2022

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 i, iий елемент ps співпадає з iим 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.

Програмний код

lab4func.sml

```
(*Task_01*)
fun only_capitals(stringList: string list) =
List.filter (fn str => Char.isUpper(String.sub(str,0))) stringList;

(*Task_02*)
fun longest_string1(stringList: string list) =
List.foldl (fn(str1, str2) =>    if (String.size(str1) > String.size(str2))
                                 then str1
                                 else str2)
           ""
           stringList;

(*Task_03*)
fun longest_string2(stringList: string list) =
List.foldl (fn(str1, str2) =>    if (String.size(str1) >= String.size(str2))
                                 then str1
                                 else str2)
           ""
           stringList;

(*Task_04*)
fun longest_string_helper f =
List.foldl (fn(str1,str2) =>    if f(String.size(str1),String.size(str2))
                                then str1
                                else str2)
           "";

val longest_string3 = longest_string_helper(fn (str1, str2) => str1 > str2);
val longest_string4 = longest_string_helper(fn (str1, str2) => str1 >= str2);

(*Task_05*)
val longest_capitalized = longest_string1 o only_capitals;

(*Task_06*)
val rev_string = String.implode o rev o String.explode;

(*Task_07*)
exception NoAnswer;
fun first_answer f list =
case list of
    [] => raise NoAnswer
    |x::xs => case f(x) of
        NONE => first_answer f xs
        |SOME result => result;
```

```
(*Task_08*)
fun all_answers f list=
    let fun find_all_answers(resultList, lst) =
        case lst of
            [] => SOME(resultList)
            |x::xs => (case f(x) of
                        NONE => NONE
                        |SOME result => find_all_answers (resultList @ result, xs))
    in
        find_all_answers([], list)
    end;

(*Task_09*)
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

(*Task_09_1*)
fun count_wildcards (p: pattern) = g (fn _ => 1) (fn _ => 0) p;

(*Task_09_2*)
fun count_wild_and_variable_lengths(p:pattern)=
    g (fn _ => 1) (String.size) p;

(*Task_09_3*)
fun count_some_var(str : string, p : pattern)=
```

```
    g (fn _ => 0) ( fn x => if String.isSubstring str x then 1 else 0) p;

(*Test_Task_10*)
fun check_pat(p: pattern)=
    let
        fun getAllStrings(p: pattern) =
            case p of
                Variable x => [x]
                |ConstructorP(_, str) => getAllStrings(str)
                |TupleP ps => List.foldl(fn(ps_elem, acc)=> getAllStrings(ps_elem)
@ acc) [] ps
                | _ => []

        fun checkDuplicates(allStr: string list) =
            case allStr of
                [] => true
                |x::xs => if (List.exists(fn str => str = x) xs)
                        then false
                        else checkDuplicates(xs)
        in
            checkDuplicates(getAllStrings(p))
        end;

(*Test_Task_11*)
fun first_match value patternList =
    let
        fun match_function (value, pattern) =
            case (value, pattern) of
                (_, Wildcard) => SOME []
                |(_, Variable s ) => SOME [(s, value)]
                |(Unit, UnitP) => SOME []
                |(Const v1, ConstP p1) =>if (v1 = p1) then SOME [] else NONE

                |(Tuple vs, TupleP ps) =>   if List.length vs = List.length ps
                                            then case all_answers
match_function(ListPair.zip(vs,ps))  of
                                                    SOME result => SOME result
                                                    |_ => NONE
                                            else NONE
                |(Constructor (s2 ,v), ConstructorP (s1, p) ) =>    if s2 = s1
                                                                    then
match_function(v,p)
                                                                    else NONE
                |(_,_) => NONE
    in
        SOME (first_answer (fn pattern => match_function (value, pattern))
patternList)
```

```
    handle NoAnswer => NONE
end;
```

Тести

lab4test.sml

```
use ("lab4func.sml");

fun test(fun_name : string, result_true, result_actual) =
if result_true = result_actual
then (fun_name, "Passed")
else (fun_name, "Failed");

(*Test_Task_01*)
print "\nTest_Task_01:\n";
(*Test_Task_01*)
print "\nTest_Task_01:\n";
test("only_capitals", ["To", "Be"], only_capitals(["To", "be", "oR", "nOt", "to",
"Be"]));
test("only_capitals", [], only_capitals(["to", "be", "or", "not", "to", "be"]));

(*Test_Task_02*)
print "\nTest_Task_02:\n";
test("longest_string1", "", longest_string1([]));
test("longest_string1", "not", longest_string1(["to", "be", "or", "not", "to",
"be"]));
test("longest_string1", "abc", longest_string1(["abc", "to", "be", "or", "not",
"to", "be"]));

(*Test_Task_03*)
print "\nTest_Task_03:\n";
test("longest_string2", "", longest_string2([]));
test("longest_string2", "not", longest_string2(["to", "be", "or", "not", "to",
"be"]));
test("longest_string2", "not", longest_string2(["abc", "to", "be", "or", "not",
"to", "be"]));

(*Test_Task_04*)
print "\nTest_Task_04:\n";
test("longest_string3", "", longest_string3([]));
test("longest_string3", "not", longest_string3(["to", "be", "or", "not", "to",
"be"]));
test("longest_string3", "abc", longest_string3(["abc", "to", "be", "or", "not",
"to", "be"]));

test("longest_string4", "", longest_string4([]));
test("longest_string4", "not", longest_string4(["to", "be", "or", "not", "to",
"be"]));
test("longest_string4", "not", longest_string4(["abc", "to", "be", "or", "not",
"to", "be"]));
```

```
(*Test_Task_05*)
print "\nTest_Task_05:\n";
test("longest_capitalized", "", longest_capitalized([]));
test("longest_capitalized", "Or", longest_capitalized(["to", "be", "Or", "not",
"to", "be"]));
test("longest_capitalized", "To", longest_capitalized(["abc", "To", "be", "or",
"not", "to", "Be"]));

(*Test_Task_06*)
print "\nTest_Task_06:\n";
test("rev_string", "nihkimyskaM", rev_string("Maksymikhin"));

(*Test_Task_07*)
print "\nTest_Task_07:\n";
test("first_answer", 4, first_answer (fn(x) => if x > 3 then SOME x else
NONE)  [1,2,3,4,5]);
test("first_answer", "c", first_answer (fn(x) => if x = "c" then SOME x else
NONE)  ["a","b","c","d"]);

(*Test_Task_08*)
print "\nTest_Task_08:\n";
test("all_answers", SOME [4, 5], all_answers (fn(x) => if x > 3 then SOME [x] else
NONE)  [4,5]);
test("all_answers", NONE , all_answers (fn(x) => if x > 3 then SOME [x] else
NONE)  [1,2,3,4,5]);
test("all_answers", SOME ["c","c","c"], all_answers (fn(x) => if x = "c" then SOME
[x] else NONE)  ["c","c","c"]);
test("all_answers", NONE , all_answers (fn(x) => if x = "c" then SOME [x] else
NONE)  ["a","c","b","c","d","c"]);

(*Test_Task_09_1*)
print "\nTest_Task_09_1:\n";
test("count_wildcards", 1 , count_wildcards Wildcard);
test("count_wildcards", 2, count_wildcards (TupleP ([Wildcard, Wildcard])));
test("count_wildcards", 3, count_wildcards (TupleP ([Variable("abc"), Wildcard,
Wildcard, ConstP(9), Wildcard])));

(*Test_Task_09_2*)
print "\nTest_Task_09_2:\n";
test("count_wild_and_variable_lengths", 2, count_wild_and_variable_lengths (TupleP
([Wildcard, Wildcard])));
test("count_wild_and_variable_lengths", 3, count_wild_and_variable_lengths (TupleP
([Variable("abc"), ConstP(9)])));
test("count_wild_and_variable_lengths", 5, count_wild_and_variable_lengths (TupleP
([Variable("abc"), Wildcard, Wildcard, ConstP(9)])));
```

```
(*Test_Task_09_3*)
print "\nTest_Task_09_3:\n";
test("count_some_var", 2, count_some_var("abc", (TupleP ([Variable("abc"),
Variable("abcd")]))));
test("count_some_var", 1, count_some_var("abc", Variable("abc")));
test("count_some_var", 2, count_some_var("abc", (TupleP ([Variable("abc"),
Variable("abc"), Variable("ab"), Wildcard]))));
test("count_some_var", 0, count_some_var("abc", ConstructorP ("abc",(Wildcard))));

(*Test_Task_10*)
print "\nTest_Task_10:\n";
test("check_pat", true, check_pat(Variable("abc")));
test("check_pat", true, check_pat((TupleP ([Variable("abc"), Variable("abd"),
Variable("abe")]))));
test("check_pat", true, check_pat((TupleP ([Variable("abc"), Variable("abd"),
Variable("abe"), ConstructorP ("abf", (Wildcard))]))));

(*Test_Task_11*)
print "\nTest_Task_11:\n";
test("first_match", SOME [("abc", Const 3)], first_match (Constructor("abc", Const
3)) ([ConstructorP("abc", Variable("abc")), Wildcard]));
test("first_match", SOME [("abc", Const 5)], first_match (Const 5)
([Variable("abc"), Wildcard, UnitP]));
test("first_match", SOME [("abcd", Unit)], first_match (Unit) ([Variable("abcd"),
Wildcard, Variable("abc")]));
```

Результати виконання тестів

```
Test_Task_01:
val it = () : unit
val it = ("only_capitals","Passed") : string * string
val it = ("only_capitals","Passed") : string * string

Test_Task_02:
val it = () : unit
val it = ("longest_string1","Passed") : string * string
val it = ("longest_string1","Passed") : string * string
val it = ("longest_string1","Passed") : string * string

Test_Task_03:
val it = () : unit
val it = ("longest_string2","Passed") : string * string
val it = ("longest_string2","Passed") : string * string
val it = ("longest_string2","Passed") : string * string

Test_Task_04:
val it = () : unit
val it = ("longest_string3","Passed") : string * string
val it = ("longest_string3","Passed") : string * string
val it = ("longest_string3","Passed") : string * string
val it = ("longest_string4","Passed") : string * string
val it = ("longest_string4","Passed") : string * string
val it = ("longest_string4","Passed") : string * string

Test_Task_05:
val it = () : unit
val it = ("longest_capitalized","Passed") : string * string
val it = ("longest_capitalized","Passed") : string * string
val it = ("longest_capitalized","Passed") : string * string

Test_Task_06:
val it = () : unit
val it = ("rev_string","Passed") : string * string

Test_Task_07:
val it = () : unit
val it = ("first_answer","Passed") : string * string
val it = ("first_answer","Passed") : string * string
```

```
Test_Task_08:
val it = () : unit
val it = ("all_answers","Passed") : string * string
val it = ("all_answers","Passed") : string * string
val it = ("all_answers","Passed") : string * string
val it = ("all_answers","Passed") : string * string

Test_Task_09_1:
val it = () : unit
val it = ("count_wildcards","Passed") : string * string
val it = ("count_wildcards","Passed") : string * string
val it = ("count_wildcards","Passed") : string * string

Test_Task_09_2:
val it = () : unit
val it = ("count_wild_and_variable_lengths","Passed") : string * string
val it = ("count_wild_and_variable_lengths","Passed") : string * string
val it = ("count_wild_and_variable_lengths","Passed") : string * string

Test_Task_09_3:
val it = () : unit
val it = ("count_some_var","Passed") : string * string
val it = ("count_some_var","Passed") : string * string
val it = ("count_some_var","Passed") : string * string
val it = ("count_some_var","Passed") : string * string

Test_Task_10:
val it = () : unit
val it = ("check_pat","Passed") : string * string
val it = ("check_pat","Passed") : string * string
val it = ("check_pat","Passed") : string * string

Test_Task_11:
val it = () : unit
val it = ("first_match","Passed") : string * string
val it = ("first_match","Passed") : string * string
val it = ("first_match","Passed") : string * string
```