## Робота №03

*Прості моделі обчислень*

Алфавіт *T* – скінченна множина символів. Слово в алфавіті *T* – довільна скінченна послідовність символів *T*. Слово *α* , яке є послідовність символів *a1, a2, …, an,* (*a1, a2, …, an ϵ T*, можливо, з повтореннями) позначається *α = a1 a2 … an.* Довжина слова *α = a1 a2 … an* – кількість символів в ньому, |α| = n.Порожнє слово – це слово, яке немає жодного символу, позначається ε, |ε| = 0*.* Множину всіх слів в алфавіті *T* позначають - *T\*.*

Слово α входить в слово β, якщо існують такі слова α1 і α2, що β = α1 α α2

Нехай алфавіт *T* складається з двох символів T = {a,b}.

* ε, a, aa, bbb, ab - приклади слів в алфавіті *T.*
* Слово ab входить в слово abbbab два раза: **ab**bbab i abbb**ab**
* Слово aa входить в слово aaabaaaa п”ять разів: **aa**abaaaa, a**aa**baaaa, aaab**aa**aa, aaaba**aa**a i aaabaa**aa**
* Слово ε (порожнє) входить в слово ab три раза: .ab, a.b i ab. (крапка позначає місце входження порожнього слова

Нормальний алгоритм Маркова *U* в алфавіті *T* – скінченна впорядкована послідовність правил виведення (підстановок) виду α –> β або α –> .β, де α, β ϵ Т\* , а символи –, > і . не належать алфавіту Т.

U = < α1 –> β1, α2 –> β2 , …, αm –> βm>

Деякі (а можливо жодної) підстановки мають вигляд αj –> .βj *.* Підстановка виду α –> .β – заключна.

Кожний алгоритм U може обробити довільне слово x в алфавіті T, перетворюючи його в інше слово в алфавіті T, котре називають *результатом обробки* і позначають U(x).Говорять, що алгоритм U застосовується до слова x

Обробка виконується послідовно, x=x0 => x1, x1 => x2, …, xn-1 => xn,і може або закінчитися або НЕ закінчитися. На кожному кроці обробки xn ϵ Т*\**знаходиться *перша по порядку* підстановка αi –> βi або αi –> .β*i* така, що αi підслово слова *xn*

* ЗНАЙШЛИ таку підстановку: саме *ліве входження* *αi* в *xn* замінюється на *βi,* утворюючи наступне словообробки *xn+1*
  + Якщо підстановка – заключна (*αi –> .βi*), то обробка завершується і *xn+1*– її результат. U(x) = xn+1.
  + Якщо підстановка – НЕ заключна (*αi –>βi*), то обробка продовжується.
* НЕ ЗНАЙШЛИ – обробка завершується і *xn* – її результат. U(x) = xn.
* Якщо процес обробки НЕ завершується ніколи, то вважається, що результат обробки U(x)- *невизначено.*
  + Один крок виконання нормального алгоритму, що перетворює слово x в слово y будемо позначати x => y, а декілька послідовних кроків - x =\*> y.

Алгоритм clearBeginOne = <ca ->. ε, cb ->. ε, ε -> c > складається з трьох правил, перші два з яких заключні. Приклади його обробки (символи \*..\* вказують місце застосування одної з підстановок)

* clearBeginOne(abb) = bb
  + \*\*abb => \*ca\*bb => bb
* clearBeginOne(ba) = b
  + \*\*ba => \*cb\*a => a
* clearBeginOne(ε) = *невизначено*
  + \*\* => \*\*c => \*\*cc => …

*Машина з необмеженими регістрами (МНР*) – простий ідеалізований комп’ютер, котрий має нескінченну кількість регістрів та програму, котру він може виконати. Кожний регістр може містити довільне натуральне число. Регістри позначаються R1, R2, …, відповідно. Машина виконує послідовно команди програми, змінюючи зміст регістрів.

*Програма машини (МР програма)* – це впорядкований список команд P = <I1, .., Ik>. Кожна команда в програмі має свій номер – порядок її розташування. Є *команди* лише чотирьох видів:

* Z(n) – замінює вміст регістру з номером n Rn на 0. Зміст інших регістрів не змінюється.
* S(n) – збільшує вміст регістра з номером n Rn на 1. Зміст інших регістрів не змінюється.
* T(m,n) – замінює вміст регістру з номером n Rn на число, що міститься в регістрі з номером m Rm. Змінює зміст лише регістру Rn.
* J(m,n,q) – команда умовного переходу, котра може змінити послідовне виконання команд програми. Якщо вміст регістрів Rn і Rm співпадають, то наступною буде виконуватися команда з номером q. Зміст всіх регістрів не змінюється.

Для *виконати обчислення* машині з необмеженими регістрами (МНР) потрібно надати програму P і послідовність натуральних чисел (a1, a2, …, ak ) – що вказує значення в перших регістрах R1, R2, …Rk  перед початком обчислення. Початкове значення всіх інших регістрів вважається рівним 0.

Програма P = <I1, .., Ik> виконується послідовно починаючи з команди з номером 1. Змінити послідовне виконання команд програми може лише команда виду J(m,n,q). Якщо номер цієї команди p і вміст регістрів Rn і Rm співпадає, то далі буде виконуватися команда з номером q, в іншому випадку наступною буде виконуватися команда з номером (p+1). Виконання програми P зупиниться, коли необхідно буде виконати команду з номером q > k. Це можливо у випадках:

* Виконується команда виду J(m,n,q), вміст регістрів Rn і Rm співпадає і наступною потрібно виконати команду з номером q > k.
* Виконується остання команда програми Ik і далі потрібно виконати команду з номером (k+1) > k.

Значення в регістрі R1  (число b1)при зупинці програми – це *результат роботи* програми. Таке виконання обчислення позначають P(a1, a2, …, ak ) = b1.

В деяких випадках машина, почавши виконувати програму P при початкових значеннях (a1, a2, …, ak ) ніколи не зупиниться. Це позначають P(a1, a2, …, ak ) = *невизначено.*

В кожній програмі P завжди використовується скінченна кількість регістрів ρ(P). Тому більшість регістрів (номера яких більше ρ(P)) буде завжди мати значення рівне 0, а всі регістри, що використовуються, утворюють скінченний список.

Програма subtraction = <Z(3), J(1,2,6), S(2), S(3), J(1,1,2), T(3,1)> обчислює функцію x-y, котра визначена для натуральних чисел x і y, x ≥ y.

* subtraction (5,2) = 3
* subtraction (2,50 = *невизначено*

Алгоритм (Algorithm) – це список підстановок (Substitution), кожна з яких є кортеж (ls, rs, e) ls – права частина підстановки, rs – ліва частина підстановки, e – признак заключної підстановки. Підстановка (ls,rs,b) застосовується до слова w в позиції i, якщо w = x++ls++y i |x| == i (0 <= i < |w|). На кожному кроці виконання НАМ застосовується сама перша підстановка в самій лівій позиції (min i).

***type*** Algorithm = [Substitution]

***type*** Substitution = (String,String,Bool)

***type*** ConfigA = (Bool, Int, String)

Виконання нормального алгоритму Маркова – це циклічний процес, на кожному кроці якого змінюється конфігурація (ConfigA) алгоритму – кортеж типу (Bool, Int, String). Кожна конфігурація (bt,st,word) включає: вказівку bt чи потрібно продовжувати застосування підстановок, лічильник st - кількість вже виконаних підстановок і слово word – результат перетворень.

Програма машини з необмеженими регістрам (Program) – це список команд (Command), кожна з яких будується одним з конструкторів Z n, S n, T n m або J n m q.

***data*** Command = Z Int | S Int | T Int Int | J Int Int Int

***deriving*** (Show, Eq)

***type*** Program = [Command]

***type*** ConfigC = (Int, Int, [Int])

Виконання програми – це циклічний процес, на кожному кроці якого змінюється конфігурація (ConfigC) машини – кортеж типу (Int, Int, [Int]). Кожна конфігурація (nm,st,reg) включає: nm номер команди, котру необхідно виконати, лічильник st - кількість вже виконаних команд і reg список значень всіх регістрів, що використовуються, – результат обчислення.

В допоміжному файлі, котрий включає визначення типів і даних для тестування, надати визначення наступних функцій.

1. Функція *isPrefix bs xs* , що перевіряє, чи являється слово bs - префіксом (початком) xs.

* isPrefix “” “abc” = True
* isPrefix “abe” “baec” = False
* isPrefix “ab” “abbb” = True

1. Функція *substitute sub i w,* котразастосовує (виконує) підстановку *sub* до слова *w* в позиції *i*: ? чи може не бути підстановки
   * substitute (“ab”,”c”, False) 2 “ioab” = “ioc”
   * substitute (“”,”ac”, True) 0 “ioab” = “acioab”
2. Функція *findPosition w sub*, що знаходить список *[(sub,i1), …, (sub,ik)]* всіх можливих місць (*i1 < … < ik*) застосування підстановки *sub* до слова *w*. Специфікація
   * findPosition “aaabaac” (“aa”,”b”,False) =

[((“aa”,”b”,False),0), [((“aa”,”b”,False),1), ((“aa”,”b”,False),4)]

* + findPosition “aaabaac” (“ca”,”b”,False) = []

1. Функція *findAll algo w*  формує список всіх можливих застосувань підстановок алгоритму *algo* до слова *w*. В списку спочатку ідуть всі можливі місця виконання першої підстановки, потім другої і т. д. Для кожної підстановки позиції впорядковані від лівої до правої.
   * findAll clearBeginOne “cab” = [((“ca”,””,True),0), ((“”,”c”,False),0),

((“”,”c”,False),1), ((“”,”c”,False),2), ((“”,”c”,False),3)]

* + findAll clearBeginOne “” = [((“”,”c”,False),0)]

1. Функція *stepA algo (bt,st,word)* реалізує один крок (застосування одної підстановки) алгоритму *algo* на конфігурації *(bt,st,word)*, створюючи нову конфігурацію.
   * stepA addEnd (True, 0, “”) = (True, 1, “c”)
   * stepA addEnd (True, 1, “c”) = (False, 2, “abb”)
2. Функція *evalA algo m word* застосовує алгоритм *algo* до слова *word*, виконуючи не більше ніж *m* підстановок. Якщо процес обчислення не завершується за *m* кроків, то результат невизначений - Nothing.
   * evalA addEnd 5 “ab” = Just “ababb”
   * evalA addEnd 5 “ababa” = Nothing
3. Функція *maximReg p*r - знаходить найбільший номер регістра, що вживається в програмі *pr.* 
   * maximReg notSignum = 2
   * maximReg subtraction = 3
4. Функція *ini pr ir*  - за програмою *pr* та значенням перших регістрів *ir*, формує початкове значення списку регістрів.

Функція *upd reg r v* - змінює значення списку регістрів *reg*, встановлюючи значення елементу списку з номером *r (r≥0)* рівним *v*. Елементи списку нумеруються (індексуються) з 0.

* ini notSignum [5] = [5,0]
* upd [5,3,2] 0 1 = [1,3,2]

1. Функція *stepC pr (nm,st,rg)* реалізує один крок виконання програми МНР - виконання однієї команди програми *pr* в конфігурації *(nm,st,rg)*.
   * stepC notSignum (1,0,[5,0]) = (2,1,[5,0])
2. Функція evalC pr mx ir - виконує програму pr на початкових даних ir, виконується не більше ніж mx кроків (команд). Якщо машина НЕ зупиниться, то результат Nothing.
   * evalC subtraction 7 [3,2] = Just 1
   * evalC subtraction 7 [2,3] = Nothing
   * evalC subtraction 6 [3,2] = Nothing

*isPrefix*  :: String -> String -> Bool

*substitute* :: Substitution -> Int -> String -> String

*findPosition* :: String -> Substitution -> [(Substitution,Int)]

*findAll* :: Algorithm -> String -> [(Substitution,Int)]

*stepA* :: Algorithm -> ConfigA -> ConfigA

*evalA* :: Algorithm -> Int -> String -> Maybe String

*maximReg* :: Program -> Int

*ini* :: Program -> [Int] -> [Int]

*upd* :: [Int] -> Int -> Int-> [Int]

*stepC* :: Program -> ConfigС -> ConfigС

*evalC* :: Program -> Int -> [Int] -> Maybe Int

Зауваження:

Назва файлу Family03.hs (Family – прізвище студента). Файл включає модуль Family03 і створюється на основі файла-заготовки HWP03.hs