## Робота №08

*Рекурсивні функції*

Розглядаються частково-визначені функції натурального аргументу.

Кожна така функція f має – k (k>0) аргументів – натуральні числа 0,1, …, N,… і результат її обчислення – теж натуральне число - f: Nk -> N**.**

Запис, f(x1, x2, …, xk) = y, традиційно, показує що функція f застосовується до аргументів-натуральних чисел x1, x2, …, xk і її результат є натуральне число y.

Загальний метод породження (побудови) обчислювальних функцій, в тому числі частково-рекурсивних функцій:

* Фіксується скінченна множина базових функцій, кожна з яких легко обчислюється.
* Фіксується декілька конструктивних операцій, кожна з яких будує нову функцію з раніше побудованих обчислювальних функцій.
  + З кожною операцією зв’язується метод обчислення значення нової функції, через значення функцій з яких вона побудована.
* Інших обчислювальних функцій не існує.

Є три *базові функції* від одного аргументу:

* z1(x1) = 0, a1(x1) = x1+ 1, (x1) = x1.

Для кожного n (n>1) є *n-базових функцій селекторів*:

* (x1,x2,…,xn) = x1, (x1,x2,…,xn) = x2, …, (x1,x2,…,xn) = xn

Задані: функція g(x1, …, xn) від n аргументів та n функцій g1(x1, …, xm), …, gn(x1, …, xm) від m аргументів. *Операція суперпозиції* (g: g1, …, gn) будує з цих n+1 функцій нову функцію f(x1, …, xm) від m аргументів.

Значення функції fв точці (a1, …, am)обчислюється наступним чином:

* Вираховуються значення b1 = g1(a1, …, am), …, bn = gn(a1, …, am)функцій g1, …, gnв точці (a1, …, am).
* Вираховується значення c = g(b1, …, bn).
* Покладається f(x1, …, xm)= c.
* Використовуючи традиційне позначення композиції функцій, можна записати

f(x1, …, xm) = g(g1(x1, …, xm), …, gn(x1, …, xm)).

*Операція примітивної рекурсії* [g,h] за двома функціями g(x1, …, xn) і h(x1, …, xn, xn+1, xn+2) *,* що мають, відповідно, n і (n+2) аргументи, будує нову функцію f(x1, …, xn, xn+1) , що має (n+1)*–*аргумент.

Для обчислення значення функції f в точці(a1, …, an, an+1) послідовно вирахувати наступні значення функції f:

* f(a1, …, an, 0) = g(a1, …, an)

f(a1, …, an, 1) = h(a1, …, an, 0, f(a1, …, an, 0))

*……………………………………………………….*

f(a1, …, an, an+1) = h(a1, …, an, an+1 - 1, f(a1, …, an, an+1 - 1))

Тобто функція f(x1, …, xn, xn+1) обраховується за рекурсивною схемою (процедурою)

* f(x1, …, xn, 0) = g(x1, …, xn)
* f(x1, …, xn, y+1) = h(x1, …, xn, y, f(x1, …, xn, y))

У випадку одного аргументу f(x1) (n = 0) або двох аргументів f(x1,x2) (n = 1) функція *f* обраховується за схемами:

* Якщо n = 0, то g(x1) = a, «функція-число» від одного аргументу, і h(x1,x2)
  + f(0 ) = a - константа
  + f(y+1) = h(y,f(y))
* Якщо n = 1, то g(x1), довільна функція від одного аргументу, і h(x1,x2,x3)
  + f(x1,0) = g(x1)
  + f(x1,y+1) = h(x1,y,f(x1,y))

Функція, що отримується з базових функцій скінченним застосуванням операцій суперпозиції та примітивної рекурсії, називається *примітивно-рекурсивною функцією*. Примітивно-рекурсивні функції всюди визначені і ефективно обчислюємі.

*Операція мінімізації* за функцією g(x1, …, xn, xn+1) з (n+1)*-*аргументами будує функцію f(x1, …, xn) з n–аргументами. Значення функція f = {g} в точці (a1, …, an)обчислюється наступним чином:

* Послідовно обчислюються значення g(a1, …, an,y)для y = 0, 1, 2,…
  + Результат *перше* y, коли g(a1, …, an,y)=0.
  + Для всіх u ≤ y g(a1, …, an,u) - визначено і g(a1, …, an,u) > 0.
* f (a1, …, an)- *невизначено*, якщо виконується одна з умов:
  + Для довільного y≥0 g(a1, …, an,y) - визначено і g(a1, …, an,y)>0.
  + Для довільного y < u g(a1, …, an,y) - визначено і g(a1, …, an,y)>0, але g(a1, …, an,u) - *невизначено*.
  + g(a1, …, an,0) – *невизначено.*
* В роботі обчислення кожної операції мінімізації f = {g,t} обмежується натуральним числом t, тобто обчислення виконуються лише для y = 0, 1, 2,…, t.

Наступні типи даних задають множину рекурсивних функцій

***type*** System = [(String,Recur)]

***data*** Recur = Zero | Succ | Sel Int Int

| Super Recur [Recur]

| Prim Recur Recur

| Mini Recur Int

| Name String ***deriving*** (Show, Eq)

Множина рекурсивних функцій (*System*) *syst = [(n1,r1), ..,(nk,rk)]* це список пар: ім»я *ni* і означення *ri* функції.

Кожна рекурсивна функція (*Recur*) це:

* Базова функція *Zero* - z1(x1), *Succ* - a1(x1) або *Sel n k* - селектор (x1,x2,…,xn).
* *Super g [g1,…,gn]* - функція, що отримана в результаті операції суперпозиції (g: g1,…,gn).
* *Prim g h* – функція, що отримана в результаті операції примітивної рекурсії [g,h].
* *Mini g t* – функція, що отримана в результаті операції мінімізації {g,t}.
* *Name nm* – посилання на функцію з іменем nm, визначення якої знаходиться в списку функції раніше.

В операціях примітивно рекурсії, котрі будують функції від одного аргументу f(x1), використовуються “функції-числа”.

*“Функція-число”* - це унарна (функція від одного аргументу) функція-константи, що завжди обчислює деяке натуральне число.

* Zero – “функція-число”, що обчислює 0.
* Якщо f – “функція-число”, що обчислює число n, то Super Succ [f] – “функція-число”, що обчислює n+1.
* Якщо пара (nm,f) - елемент множини рекурсивних функцій (System) і f – “функція-число”, то (Name nm) – “функція-число”.
* Інших “функцій-чисел” немає.

Арність (ранг, кількість аргументів) r(f) рекурсивної функції f обчислюється за правилами:

* Ранг базових функцій Zero, Succ, Sel n k – дорівнює 1.
* Ранг результату суперпозиції Super b al - це ранг одної з функцій-аргументів r(head al).
* Результат примітивної рекурсії Prim i st - r(st)-1.
* Результат операції мінімізації Mini b - r(b) -1.
* Ранг Name f – це ранг означення функції r(f).

Існує багато формальних методів, котрі задають синтаксис мов: контекстно-вільні граматики (використовувалися раніше), форми Наура-Бекуса, синтаксичні діаграми і т.і. Для опису синтаксису системи рекурсивних функцій буде використаємо розширення форми Наура-Бекуса, котра в свою чергу є розширення контекстно-вільних граматик.

В описі мови використовуються термінали – це символи алфавіту – і нетермінали, котрі позначають певні синтаксичні конструкції мови.

Термінали позначають самі себе і записуються в лапках – *‘a’, ‘n’, ‘=’, ‘\n’*. *‘eos’* – особливий термінал, котрий позначає кінець слова (end Of String).

Нетермінали – це слова в кутових дужках - *<sign>*, *<integer>*, *<digit>*.

Опис мови – це скінченна множина правил виду:

*нетермінал* ::= *правило*

Правило - слово з терміналів, нетерміналів і метасимволів, котре описує як побудувати всі слова синтаксичної конструкції *нетермінал.*

Наступні правила описують цілі десяткові числа <integer>.

*<digit> ::= ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’*

*<integer> ::= <digit> {<digit>}*

В цих правилах вживаються метасимволи:

* *a1 | a2* - вибір одного з варіантів a1 або a2
* *{a1}* – конструкцію a1 в середині {} потрібно використати 0 або багато разів

Використання метасимволів {} надає можливостей одним правилом описати нескінченну

множину слів.

Наступна множина правил описує синтаксис системи рекурсивних функцій

*<digit> ::= ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’*

*<integer> ::= <digit> {<digit>}*

*<letter> ::= ‘a’| … \ ‘w’ | ‘A’ | … | ‘W’*

*<iden> ::= <letter> {<digit> | <letter> }*

*<recur> ::= <base> | <super> | <prim> | <mini>*

*<base> ::= ”a1” | “z1” | ‘s’<digit><digit> | <iden>*

*<super> ::= ‘(‘ <recur> ‘:’ <recur> {‘,’ <recur>} ‘)’*

*<prim> ::= ‘[‘ <recur> ‘,’ <recur> ‘]’*

*<mini> := ‘{‘ <recur> ‘,’ <integer> ‘}’*

*<system> ::= { <iden> ‘=’ <recur> ‘;’ } ‘eos’*

При заданні системи рекурсивних функцій дозволяється використовувати різні символи проміжку (проміжок, табуляцію, перехід на новий рядок – isSpace) в довільному місці крім ідентифікаторів та чисел. Ця особливість не відображена в множині правил тому, що ці символи можна просто вилучити скориставшись функцією *filter* до початку синтаксичного аналізу.

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

1. Функція *isNumbConst syst f*, котра перевіряє що *f* - є “функція-число” в системі рекурсивних функцій *syst*.
   * isNumbCons syst1 Succ = False
   * isNumbCons syst1 (Name “const0”) = True
2. Функція *evRank syst f*, що обчислює ранг (арність) функції *f* в системі *syst*.
   * evRank syst1 (Super Succ [Super Succ [Zero]]) = 1
   * evRank syst1 (Name “addition”) = 2
3. Функція *isNames syst*, котра перевіряє що імена усіх функцій в системі *sys*t визначені і вірно використовуються. Система syst = [(n1,f1), …, (nk,fk)] має вірні імена, якщо жодне ім»я не об»являється двічі і в означенні fi імені ni використовуються лише імена n1, n2, … ni-1.
   * isNames syst1 = True
   * isNames syst2 = False
4. Функція *isRecur syst f*, котра перевіряє, що означення функції *f* в системі *syst* немає помилок (вірне).
   * isRecur syst1 (Mini (Name “subtractionAbs3”) 100) = True
   * isRecur syst1 (Mini (Name “substractionAbs3”) 100) = False
5. Функція *eval syst f vl*, що обчислює примітивно рекурсивну функцію *f* в системі *syst* на наборі даних *vl*.
   * eval syst1 (Name “addition”) [6, 34] = 40
   * eval syst1 (Super Succ [Super Succ [Zero]]) [60] = 2
6. Функція *evalPart syst f vl*, котра обчислює частково-рекурсивну функцію (не всюди визначену) *f* в системі *syst* на наборі даних *vl*. Якщо результат функції *невизначено*, то повертається значення Nothing.
   * evalPart syst1 (Name “subtractionPart”) [6,3] = Just 3
   * evalPart syst1 (Name “subtractionPart”) [6, 34] = Nothing
7. Функція *parseRec str*  виконує синтаксичний аналіз рядка *str*, розпізнаючи набір рекурсивних функцій. (Можна використати модуль Parsec).
   * parseRec sysStr2 = Nothing
   * parserRec systStr1 == Just syst1 = True

*isNumbCons* :: System -> Recur -> Bool

*evRank*  :: System -> Recur -> Int

*isNames* :: System -> Bool

*isRecur* :: System -> Recur -> Bool

*eval*  :: System -> Recur -> [Int] -> Int

*evalPart*  :: System -> Recur -> [Int] -> Maybe In

*parseRec*  :: String -> Maybe System

Зауваження:

Назва файлу Family08.hs (Family – прізвище студента). Файл включає модуль Family08 і створюється на основі файла-заготовки HWP08.hs