Програмски парадигми

Домашна задача 1

Стефан Милев | 206055

1.

Дефинирани се предикати even и odd кои се враќаат yes кога листата има парен и непарен број на елементи во неа, соодветно. Дефинирани се така што се повикуваат рекурзивно еден со друг, се додека не стигне до основниот случај, каде што празна листа има парен број на елементи.

Дефиниран е и предикатот neparen_palindrom, кој враќа yes кога листата има непарен број на елементи и е палиндром. Проверката за палиндром се врши преку предикатот reverse, кој враќа yes кога листата во првиот аргумент е обратната листа од листата во вториот аргумент.

2.

Дефинирани се предикатите:

- is_length - враќа yes кога листата во првиот аргумент има должина еднаква на вториот аргумент.
- sub_array - во листата од третиот аргумент се поставуваат првите елементи од листата од првиот аргумент, и тоа толку елементи колку што е бројот предаден како втор аргумент.
- sub_arrays - го повикува предикатот sub_array за секој елемент од листата.
- count - го враќа бројот на елементите еднакви на вториот аргумент во листата од првиот аргумент.
- max - враќа yes доколку третиот аргумент е поголемиот од првите два.
- max_count - во вториот аргумент го враќа бројот на појавувања на најчестата подниза преку итерирање рекурзивно на сите поднизи од првиот аргумент и брои колку пати се појавува секоја во листата од поднизи.
- find_with_count - третиот аргумент станува најчестата подниза преку рекурзивно изминување на сите поднизи.
- naj_podniza - во третиот аргумент ја враќа поднизата со должина еднаква на вториот аргумент која се појавува во листата од првиот аргумент.

3.

Дефиниран е предикатот proveri кој враќа no доколку листата нема најмалку два елементи (не е дефиниран тој случај, па се смета за неточен). Доколку има два елементи во листата, тогаш враќа yes доколку вториот елемент е поголем од првиот. Инаку, доколку листата има најмалку 3 елементи, тогаш враќа yes доколку секој елемент е најизменично поголем, па помал, па поголем, и така натаму, при што се повикува предикатот рекурзивно со тргање на првите два елементи на секој повик.

4.

Дефиниран е предикатот element кој враќа yes кога првиот елемент во листата е елементот во вториот аргумент на предикатот. Предикатот permutation зема еден по еден елемент користејќи го element предикатот, па го брише елементот од листата и се повикува рекурзивно со останатите елементи се додека има, а потоа прави findall на сите пермутации за да се соберат во една листа. Предикатот permutacii ги генерира сите можни пермутации на листата дадена во првиот аргумент и ги поставува во вториот аргумент.

5.

Собирање

Предикатот sobiranje работи така што двата бинарни броеви се превртени, односно запишани обратно, па се повикува предикатот add, кој е дефиниран за секој случај каде што може да се најдат преостанати битови за собирање. R е збирот на дадените битови со пренесеното од претходниот збир (или carry). Ова потоа се извршува во предикатот addCommon, каде што се зема остатокот при делење со 2 од збирот за резултантниот бит, а потоа од збирот се калкулира следниот пренесен бит (carry).

Одземање

Логиката за одземање е слична како таа за собирање, со разлика во тоа што нулите се вадат на лево од резултатот и пренесениот (carry) бит се рачуна како посебен предикат кој што кажува дека, ако збирот е -1, тогаш carry битот ќе биде -1, а во секој друг случај, 0.

Множење

Множењето е имплементирано како скратено собирање. Вториот број се превртува и се прави рекурзија за да се најде бит во вториот број кој толку пати ќе се изврши собирање експоненцијално на левиот број, односно 2^n, каде што n е локацијата на битот 1. Основниот случај на рекурзијата за собирањето е [0].

Делење

Делењето е имплементирано, така што, се додека левиот број е поголем од десниот, десниот број се одзема од левиот и притоа да го додава битот [1] на резултатот кој на почеток е сетиран како [0].

6.

Предикатот list_multiply е за множење на две листи, елемент по елемент. Предикатот list_product множи две листи и го собира нивниот збир. Предикатот list_calc го множи редот со секој друг ред од матрицата и резултатот го собира во резултантната листа. Предикатот presmetaj го повикува list_calc за секој ред од матрицата за да се добие матрицата помножена со транспонираната матрица.

7.

Предикатот prepend го поставува елементот во вториот аргумент на почеток на листата од третиот аргумент во форма [2|1]. Предикатот should_go_left е точен кога вториот аргумент треба да се постави лево од првиот во сортираната листа. Предикатот should_go_left_same_len е специјален случај за проверката на редоследот за две листи кои имаат иста должина, и притоа се проверува дали првиот елемент е поголем. Предикатот sort го повикува bubble sort алгоритамот преку предикатот sort_helper, кој извршува една итерација на алгоритамот, при што секој пар на листи ги заменува доколку е потребно. Во третиот аргумент на предикатот sort се наоѓа бројот на итерации на bubble sort алгоритамот. Предикатот unique е за филтрирање на идентични листи, т.е. листи со еднаква должина и елементи на секоја позиција, и предикатот transform го повикува предикатот sort и има вкупно итерации колку што е должината на листата, а потоа го користи предикатот unique за да се извадат сите идентични листи од резултатот.

8.

Дефиниран е предикатот delete кој ги опфаќа сите случаи за бришење на елемент од листата. Првиот случај е за доколку следниот елемент е листа, при што се повикува рекурзивно на елементите од подлистата, па продолжува понатаму. Вториот случај е за кога следниот елемент не е листа, и кога не е дел од третиот аргумент, кој е листа во која се чуваат елементите кои се веќе изминати. Се додава следниот елемент на листата на изминати елементи, па продолжува натаму. Третиот случај е доколку е најден веќе изминат елемент, се брише елементот од изминатите и не се додава во резултантната листа. Последниот случај е базичниот за крај на рекурзијата, кога нема веќе елементи за изминување. Потоа, предикатот brisi_sekoe_vtoro го повикува предикатот delete.