**LABORATOR 2**

fmod NAT1 is

sort Nat .

op 0 : -> Nat .

op s\_ : Nat -> Nat .

op \_+\_ : Nat Nat -> Nat .

vars X Y : Nat .

eq X + 0 = X .

eq X + s Y = s (X + Y) .

endfm

fmod FIBO is

extending NAT1 .

op fib : Nat -> Nat .

var X : Nat .

eq fib(0) = 0 .

eq fib(s 0) = s 0 .

eq fib(s s X) = fib(s X) + fib(X) .

\*\*\*( al n-lea termen din

sirul lui Fibonacci )

endfm

fmod FIBONACCI is

extending NAT .

op fib : Nat -> Nat .

var X : Nat .

eq fib(0) = 0 .

eq fib(1) = 1 .

eq fib(s s X) = fib(s X) + fib(X) .

\*\*\*( al n-lea termen din sirul

lui Fibonacci, cu numerele

naturale predefinite )

endfm

fmod M3 is

sort Nat .

op 0 : -> Nat .

op s\_ : Nat -> Nat .

op m3 : Nat -> Bool .

\*\*\*( testeaza daca argumentul sau

este multiplu de 3, folosind

doar operatiile 0 si succesor )

\*\*\*( A se vedea operatia

\_divides\_ din modulul

NAT predefinit. )

var X : Nat .

eq m3(0) = true .

eq m3(s 0) = false .

eq m3(s s 0) = false .

eq m3(s s s X) = m3(X) .

endfm

fmod Z2 is

\*\*\*( specificatie pentru clasele

de resturi modulo 2 )

sort Nat .

op 0 : -> Nat .

op s\_ : Nat -> Nat .

var X : Nat .

eq s s X = X .

endfm

fmod FP is

sort Nat .

op 0 : -> Nat .

op s\_ : Nat -> Nat .

ops \_+\_ \_\*\_ : Nat Nat -> Nat .

op \_! : Nat -> Nat . \*\*\*> factorialul

op \_^\_ : Nat Nat -> Nat .

\*\*\*( ridicarea la putere, operatie care, in

modulul predefinit NAT, este predefinita )

vars X Y : Nat .

eq X + 0 = X .

eq X + s Y = s (X + Y) .

eq X \* 0 = 0 .

eq X \* s Y = (X \* Y) + X .

eq 0 ! = s 0 .

eq (s X) ! = (X !) \* (s X) .

eq X ^ 0 = s 0 .

eq X ^ (s Y) = (X ^ Y) \* X .

endfm

fmod FACTORIAL is

extending NAT .

op \_! : Nat -> Nat . \*\*\*> factorialul

var X : Nat .

eq 0 ! = 1 .

eq (s X) ! = (X !) \* (s X) .

endfm

fmod ERONAT is

\*\*\*> cum NU se scrie o specificatie in Maude

sort Nat .

op 0 : -> Nat .

op s\_ : Nat -> Nat .

op \_+\_ : Nat -> Nat .

vars X Y : Nat .

eq X + 0 = X .

eq X + s Y = s (X+Y) .

eq s X + s Y = s (X+Y) . \*\*\*( ecuatie prin

adaugarea careia se pierde confluenta rescrierii )

eq s X = s s s X . \*\*\*( ecuatie prin adaugarea

careia se pierde terminarea rescrierii )

endfm

**LABORATOR 3**

fmod NATP1 is

sort Nat .

op 0 : -> Nat .

ops s\_ p\_ : Nat -> Nat .

\*\*\*( numerele naturale, cu 0,

succesor si predecesor )

var X : Nat .

eq p s X = X .

eq s p X = X .

endfm

red p 0 .

red p p p 0 .

red p s 0 .

red p s s p s 0 .

red s p 0 .

red p p p s p s s 0 .

fmod NATP2-1 is

sorts NzNat Nat .

subsort NzNat < Nat .

op 0 : -> Nat .

op s\_ : Nat -> NzNat .

op p\_ : NzNat -> Nat .

var X : Nat .

var Y : NzNat .

eq p s X = X .

eq s p Y = Y .

endfm

red p 0 .

red p p p 0 .

red p s 0 .

red p s s p s 0 .

red s p 0 .

red p p p s p s s 0 .

fmod NATP2-2 is

sorts Zero NzNat Nat .

subsorts Zero NzNat < Nat .

op 0 : -> Zero .

op s\_ : Nat -> NzNat .

op p\_ : NzNat -> Nat .

var X : Nat .

var Y : NzNat .

eq p s X = X .

eq s p Y = Y .

endfm

red p 0 .

red p p p 0 .

red p s 0 .

red p s s p s 0 .

red s p 0 .

red p p p s p s s 0 .

fmod NATP3-1 is

sorts NzNat Nat Eroare NatEr .

subsort NzNat < Nat .

subsorts Nat Eroare < NatEr .

op 0 : -> Nat .

op s\_ : Nat -> NzNat .

op p\_ : NzNat -> Nat .

op s\_ : NatEr -> NatEr .

op p\_ : NatEr -> NatEr .

op eroare : -> Eroare .

var X : Nat .

var Y : NzNat .

eq p s X = X .

eq s p Y = Y .

eq p 0 = eroare .

eq p eroare = eroare .

eq s eroare = eroare .

endfm

red p 0 .

red p p p 0 .

red p s 0 .

red p s s p s 0 .

red s p 0 .

red p p p s p s s 0 .

fmod NATP3-2 is

sorts Zero NzNat Nat Eroare NatEr .

subsorts Zero NzNat < Nat .

subsorts Nat Eroare < NatEr .

op 0 : -> Zero .

op s\_ : Nat -> NzNat .

op p\_ : NzNat -> Nat .

op s\_ : NatEr -> NatEr .

op p\_ : NatEr -> NatEr .

op p\_ : Zero -> Eroare .

op eroare : -> Eroare .

var X : Nat .

var Y : NzNat .

eq p s X = X .

eq s p Y = Y .

eq p 0 = eroare .

eq p eroare = eroare .

eq s eroare = eroare .

endfm

red p 0 .

red p p p 0 .

red p s 0 .

red p s s p s 0 .

red s p 0 .

red p p p s p s s 0 .

fmod NATPRED is

extending NAT .

sorts Eroare NatEr .

subsorts Nat Eroare < NatEr .

op p\_ : NzNat -> Nat .

op p\_ : NatEr -> NatEr .

op eroare : -> Eroare .

\*\*\*( predecesorul pentru

numerele naturale predefinite )

var X : Nat .

var Y : NzNat .

eq p s X = X .

eq s p Y = Y .

eq p 0 = eroare .

eq p eroare = eroare .

eq s eroare = eroare .

endfm

red p 0 .

red p p p 0 .

red p s 0 .

red p 1 .

red p 10000 .

red p s s p s 0 .

red s p 0 .

red p p p s p s s 0 .

fmod PERECHI is

protecting NAT .

sort Pereche .

op (\_,\_) : Nat Nat -> Pereche .

op \_+\_ : Pereche Pereche -> Pereche .

\*\*\*> suma pe componente

op \_\*\_ : Pereche Pereche -> Nat .

\*\*\*> produsul scalar

op \_<=\_ : Pereche Pereche -> Bool .

\*\*\*> ordinea lexicografica

vars X Y X1 Y1 : Nat .

eq (X , Y) + (X1 , Y1) = (X + X1 , Y + Y1) .

eq (X , Y) \* (X1 , Y1) = X \* X1 + Y \* Y1 .

eq (X , Y) <= (X1 , Y1) = (X < X1) or ((X == X1) and (Y <= Y1)) .

endfm

fmod LISTE-NOTOK is

protecting NAT .

sort Lista .

subsort Nat < Lista .

op nil : -> Lista .

\*\*\*> lista vida

op \_\_ : Lista Lista -> Lista [assoc] .

\*\*\*> concatenarea

op lungime : Lista -> Nat .

var X : Nat .

var L : Lista .

eq L nil = L .

eq nil L = L .

eq lungime(nil) = 0 .

eq lungime(X L) = s lungime(L) .

\*\*\*( Nu functioneaza cum trebuie, pentru ca

nu stie sa rescrie o lista in ea insasi

concatenata cu lista vida (nil). Pentru aceasta,

nil trebuie declarata element neutru la concatenare,

ca in modulul urmator. )

endfm

fmod LISTE-OK is

protecting NAT .

sort Lista .

subsort Nat < Lista .

op nil : -> Lista .

\*\*\*> lista vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

\*\*\*> concatenarea

op lungime : Lista -> Nat .

var X : Nat .

var L : Lista .

eq lungime(nil) = 0 .

eq lungime(X L) = s lungime(L) .

endfm

fmod LISTE is

\*\*\*> mai multe operatii cu liste

extending NAT .

sorts Infinit NatInf Lista .

subsorts Nat Infinit < NatInf .

subsort Nat < Lista .

op infinit : -> Infinit .

op nil : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op lungime : Lista -> Nat .

\*\*\*> lungimea listei (numarul de elemente)

op suma : Lista -> Nat .

\*\*\*> suma elementelor listei

op indicipari : Lista -> Lista .

\*\*\*> sublista elementelor de indici pari

\*\*\*> (numarati de la 1, din capul listei)

op elempare : Lista -> Lista .

\*\*\*> sublista elementelor pare

op maxim : Nat Nat -> Nat .

\*\*\*> maximul a doua numere naturale

\*\*\*> exista max predefinit in NAT

op maxlista : Lista -> Nat .

\*\*\*> maximul dintr-o lista

op minim : Nat Nat -> Nat .

\*\*\*> minimul a doua numere naturale

\*\*\*> exista min predefinit in NAT

op minlista : Lista -> NatInf .

\*\*\*> minimul dintr-o lista

op nucifre : Lista -> Lista .

\*\*\*> selecteaza elementele care nu sunt cifre

op eordcresc : Lista -> Bool .

\*\*\*> testeaza daca lista e ordonata crescator

op lm3 : Nat -> Lista .

\*\*\*> lm3(N) = lista primilor N multipli pozitivi de 3

vars X Y : Nat .

var L : Lista .

eq lungime(nil) = 0 .

eq lungime(X L) = s lungime(L) .

eq suma(nil) = 0 .

eq suma(X L) = X + suma(L) .

eq indicipari(nil) = nil .

eq indicipari(X) = nil .

eq indicipari(X Y L) = Y indicipari(L) .

eq elempare(nil) = nil .

ceq elempare(X L) = X elempare(L) if 2 divides X .

ceq elempare(X L) = elempare(L) if not (2 divides X) .

ceq maxim(X,Y) = Y if X <= Y .

ceq maxim(X,Y) = X if X > Y .

eq maxlista(nil) = 0 .

eq maxlista(X L) = maxim(X,maxlista(L)) .

ceq minim(X,Y) = Y if X >= Y .

ceq minim(X,Y) = X if X < Y .

eq minlista(nil) = infinit .

eq minlista(X) = X .

eq minlista(X Y L) = minim(X,minlista(Y L)) .

eq nucifre(nil) = nil .

ceq nucifre(X L) = X nucifre(L) if X >= 10 .

ceq nucifre(X L) = nucifre(L) if X < 10 .

eq eordcresc(nil) = true .

eq eordcresc(X) = true .

eq eordcresc(X Y L) = (X <= Y) and eordcresc(Y L) .

eq lm3(0) = nil .

eq lm3(s X) = lm3(X) (3 \* (s X)) .

endfm

fmod SIR-FIBONACCI is

protecting NAT .

sort Lista .

subsort Nat < Lista .

op nil : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op sirfib : Nat -> Lista .

\*\*\*> genereaza sirul lui Fibonacci pana la un indice dat

op adauga : Lista -> Lista .

\*\*\*( adauga la o lista de lungime cel putin

2 suma ultimelor sale doua elemente )

vars X Y : Nat .

var L : Lista .

eq adauga(L X Y) = L X Y (X + Y) .

eq sirfib(0) = 0 .

eq sirfib(1) = 0 1 .

eq sirfib(s s X) = adauga(sirfib(s X)) .

endfm

fmod TEMA1 is \*\*\*> 15 puncte

\*\*\*( Definiti scaderea pe numerele naturale

definite cu specificatia lui Lawvere, precum

si pe numerele naturale predefinite. Folositi

un sort de eroare in ambele cazuri. )

\*\*\*( Aduceti aceste module la laboratorul de

saptamana viitoare. La fel pentru toate temele

care vor urma. Nu se vor puncta temele aduse cu

cel putin o saptamana intarziere. Se acorda punctaj

numai pentru temele aduse in saptamana imediat

urmatoare celei in care au fost date. )

endfm

fmod TEMA2 is \*\*\*> 30 puncte

\*\*\*( Definiti in Maude o specificatie pentru

numerele intregi, cu operatiile: 0, s (succesor),

p (predecesor), +, - unar si binar, \*. )

endfm

fmod TEMA3 is \*\*\*> 15 puncte

\*\*\*( Sa se creeze un modul pentru stive, cu

operatiile pop si push. Acelasi lucru pentru cozi. )

endfm

**LABORATOR 4**

fmod TEMA1-1 is

sorts Nat Eroare NatEr .

subsorts Nat Eroare < NatEr .

op 0 : -> Nat .

op s\_ : Nat -> Nat .

op \_-\_ : Nat Nat -> NatEr .

op eroare : -> Eroare .

vars X Y : Nat .

eq X - 0 = X .

eq s X - s Y = X - Y .

eq 0 - s Y = eroare .

\*\*\*( Pentru a se putea evalua expresii compuse

(cu mai multe operatii) in care unii subtermeni

se reduc la eroare, se pot declara urmatoarele,

si adauga ecuatiile de mai jos:

op s\_ : Eroare -> Eroare .

op s\_ : NatEr -> NatEr .

op \_-\_ : NatEr Eroare -> Eroare .

op \_-\_ : Eroare NatEr -> Eroare .

op \_-\_ : NatEr NatEr -> NatEr .

var E : NatEr .

eq s eroare = eroare .

eq E - eroare = eroare .

eq eroare - E = eroare . )

endfm

fmod TEMA1-2 is

extending NAT .

sorts Eroare NatEr .

subsorts Nat Eroare < NatEr .

op \_-\_ : Nat Nat -> NatEr .

op eroare : -> Eroare .

vars X Y : Nat .

eq X - 0 = X .

eq s X - s Y = X - Y .

eq 0 - s Y = eroare .

endfm

fmod TEMA2 is

sort Int .

op 0 : -> Int .

ops s\_ p\_ : Int -> Int .

op -\_ : Int -> Int .

ops (\_+\_) (\_\*\_) (\_-\_) : Int Int -> Int .

vars X Y : Int .

eq p s X = X .

eq s p X = X .

eq X + 0 = X .

eq X + s Y = s (X + Y) .

eq X + p Y = p (X + Y) .

eq - 0 = 0 .

eq - - X = X .

eq - s X = p (- X) .

eq - p X = s (- X) .

eq X - 0 = X .

eq X - s Y = p (X - Y) .

eq X - p Y = s (X - Y) .

eq X \* 0 = 0 .

eq X \* s Y = (X \* Y) + X .

eq X \* p Y = (X \* Y) - X .

endfm

fmod TEMA3-STIVE is

protecting NAT .

sort Stiva .

subsort Nat < Stiva .

op nil : -> Stiva .

op \_\_ : Stiva Stiva -> Stiva [assoc id: nil] .

op push : Nat Stiva -> Stiva .

op pop : Stiva -> Nat .

op popstiva : Stiva -> Stiva .

var X : Nat .

var S : Stiva .

eq push(X,S) = X S .

eq pop(X S) = X .

eq popstiva(X S) = S .

endfm

fmod TEMA3-COZI is

protecting NAT .

sort Coada .

subsort Nat < Coada .

op nil : -> Coada .

op \_\_ : Coada Coada -> Coada [assoc id: nil] .

op push : Nat Coada -> Coada .

op pop : Coada -> Nat .

op popcoada : Coada -> Coada .

var X : Nat .

var C : Coada .

eq push(X,C) = C X .

eq pop(X C) = X .

eq popcoada(X C) = C .

endfm

fmod LISTE2 is

\*\*\*> alte operatii cu liste

extending NAT .

sort Lista .

subsort Nat < Lista .

op nil : -> Lista . \*\*\*> lista vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

\*\*\*> concatenarea listelor

op sumaprec : Lista -> Lista .

\*\*\*( selecteaza elementele egale cu suma

celor ce le preceda in lista )

op aux : Nat Lista -> Lista .

\*\*\*( operatie auxiliara necesara pentru

scrierea operatiei sumaprec )

op \_apartine\_ : Nat Lista -> Bool .

\*\*\*( determina daca un element apartine

unei liste )

op nraparitii : Nat Lista -> Nat .

\*\*\*( calculeaza numarul aparitiilor

unui element intr-o lista )

op nth : NzNat Lista -> Nat .

\*\*\*> al n-lea element dintr-o lista

op inversa : Lista -> Lista .

\*\*\*> inverseaza o lista

op sterge : Nat Lista -> Lista .

\*\*\*> sterge prima aparitie a unui element intr-o lista

op stergetot : Nat Lista -> Lista .

\*\*\*> sterge toate aparitiile unui element intr-o lista

ops elimdup elimdup2 : Lista -> Lista .

\*\*\*> fiecare elimina duplicatele dintr-o lista

\*\*\*> elimdup pastreaza ultima pozitie a fiecarui element duplicat

\*\*\*> elimdup2 pastreaza prima pozitie a fiecarui element duplicat

op \_lex\_ : Lista Lista -> Bool .

\*\*\*> ordinea lexicografica

op listm7 : Nat -> Lista .

\*\*\*( genereaza lista multiplilor naturali de 7

mai mici sau egali cu un numar natural dat )

vars X Y : Nat .

vars L M : Lista .

eq sumaprec(L) = aux(0,L) .

eq aux(X,nil) = nil .

eq aux(X,X L) = X aux(X + X,L) .

ceq aux(X,Y L) = aux(X + Y,L) if X =/= Y .

eq X apartine nil = false .

eq X apartine (X L) = true .

ceq X apartine (Y L) = X apartine L if X =/= Y .

eq nraparitii(X,nil) = 0 .

eq nraparitii(X,X L) = s nraparitii(X,L) .

ceq nraparitii(X,Y L) = nraparitii(X,L) if X =/= Y .

eq nth(1,X L) = X .

eq nth(s s Y,X L) = nth(s Y,L) .

eq inversa(nil) = nil .

eq inversa(X L) = inversa(L) X .

eq sterge(X,nil) = nil .

eq sterge(X,X L) = L .

ceq sterge(X,Y L) = Y sterge(X,L) if X =/= Y .

eq stergetot(X,nil) = nil .

eq stergetot(X,X L) = stergetot(X,L) .

ceq stergetot(X,Y L) = Y stergetot(X,L) if X =/= Y .

eq elimdup(nil) = nil .

ceq elimdup(X L) = X elimdup(L) if not (X apartine L) .

ceq elimdup(X L) = elimdup(L) if X apartine L .

eq elimdup2(nil) = nil .

eq elimdup2(X L) = X elimdup2(stergetot(X,L)) .

eq nil lex L = true .

eq (X L) lex nil = false .

eq (X L) lex (X M) = L lex M .

ceq (X L) lex (Y M) = true if X < Y .

ceq (X L) lex (Y M) = false if X > Y .

eq listm7(0) = 0 .

ceq listm7(s X) = listm7(X) (s X) if 7 divides (s X) .

ceq listm7(s X) = listm7(X) if not (7 divides (s X)) .

endfm

fmod TEMA4 is \*\*\*> 40 puncte

\*\*\*( Completati modulul pentru numere intregi din

TEMA2 cu operatiile abs (modulul), div (catul

impartirii intregi), mod (restul impartirii intregi)

si cmmdc (cel mai mare divizor comun), precum si

relatiile <, <=, >, >=. Stim ca relatiile se definesc

ca operatii de sort rezultat Bool (sortul boolean).

Cat despre operatiile de mai sus, probabil ca este de

preferat sa fie definite mai intai pe numerele naturale,

adica pe intregii nenegativi, si apoi pe toti intregii.

Cel mai mare divizor comun poate fi calculat cu

algoritmul lui Euclid. )

\*\*\*( Aduceti acest modul la laboratorul de

saptamana viitoare. La fel pentru toate temele

care vor urma. Nu se vor puncta temele aduse cu

cel putin o saptamana intarziere. Se acorda punctaj

numai pentru temele aduse in decurs de o saptamana

din ziua in care au fost date. )

endfm

fmod TEMA5 is \*\*\*> 20 puncte

\*\*\*( Scrieti un modul care sa importe modulul predefinit

NAT si sa contina o operatie definita pe Nat si cu valori

booleene care sa determine daca argumentul sau este prim. )

endfm

**LABORATOR 5**

fmod TEMA4 is

sorts Zero NzNat Nat Neg NzInt Int .

subsort Zero < Nat .

subsorts NzNat < Nat NzInt < Int .

subsort Neg < NzInt .

op 0 : -> Zero .

ops s\_ p\_ : Int -> Int .

op s\_ : Nat -> NzNat .

op p\_ : NzNat -> Nat .

op p\_ : Zero -> Neg .

op p\_ : Neg -> Neg .

ops (\_+\_) (\_\*\_) (\_-\_) : Int Int -> Int .

ops (\_+\_) (\_\*\_) : Nat Nat -> Nat .

op \_+\_ : NzNat Nat -> NzNat .

op \_+\_ : Nat NzNat -> NzNat .

op \_\*\_ : NzInt NzInt -> NzInt .

op \_\*\_ : NzNat NzNat -> NzNat .

op \_\*\_ : Zero Int -> Zero .

op \_\*\_ : Int Zero -> Zero .

op -\_ : Int -> Int .

op -\_ : NzNat -> Neg .

op -\_ : Neg -> NzNat .

op abs : Int -> Nat .

op abs : NzInt -> NzNat .

ops (\_<=\_) (\_<\_) (\_>=\_) (\_>\_) : Int Int -> Bool .

op \_div\_ : Nat NzNat -> Nat .

op \_div\_ : Int NzInt -> Int .

op \_mod\_ : Int NzInt -> Nat .

op cmmdc : Int Int -> Nat .

vars X Y : Int .

vars M M1 : Nat .

var N : Neg .

var P : NzNat .

eq p s X = X .

eq s p X = X .

eq X + 0 = X .

eq X + s Y = s (X + Y) .

eq X + p Y = p (X + Y) .

eq - 0 = 0 .

eq - - X = X .

eq - s X = p (- X) .

eq - p X = s (- X) .

eq X - 0 = X .

eq X - s Y = p (X - Y) .

eq X - p Y = s (X - Y) .

eq X \* 0 = 0 .

eq X \* s Y = (X \* Y) + X .

eq X \* p Y = (X \* Y) - X .

eq abs(M) = M .

eq abs(N) = - N .

eq 0 <= M = true .

eq P <= 0 = false .

eq - M <= - M1 = M1 <= M .

eq N <= M = true .

eq M <= N = false .

eq s X <= s Y = X <= Y .

eq p X <= p Y = X <= Y .

eq X < Y = X <= Y and X =/= Y .

eq X >= Y = Y <= X .

eq X > Y = Y < X .

ceq M div P = 0 if M < P .

ceq M div P = s((M - P) div P) if P <= M .

ceq M mod P = M if M < P .

ceq M mod P = (M - P) mod P if P <= M .

eq cmmdc(0,M) = M .

eq cmmdc(P,M) = cmmdc(M mod P,P) .

ceq cmmdc(X,Y) = cmmdc(abs(X),abs(Y)) if X < 0 or Y < 0 .

eq X mod N = X mod (- N) .

ceq N mod P = 0 if (- N) mod P == 0 .

ceq N mod P = P - ((- N) mod P) if (- N) mod P =/= 0 .

eq X div N = - (X div (- N)) .

ceq N div P = - ((- N) div P) if (- N) mod P == 0 .

ceq N div P = p (- ((- N) div P)) if (- N) mod P =/= 0 .

endfm

fmod TEMA5 is

protecting NAT .

op nedivizibil : NzNat Nat -> Bool .

\*\*\*( nedivizibil(Z,X) = true daca si numai daca X nu se divide cu niciunul

dintre numerele naturale cuprinse intre Z si X inclusiv, ceea ce este echivalent

cu faptul ca X nu se divide cu niciunul dintre numerele naturale cuprinse intre

Z si radacina patrata a lui X inclusiv )

op prim : Nat -> Bool .

\*\*\*> testeaza daca argumentul sau este prim

var X : Nat .

var Z : NzNat .

ceq nedivizibil(Z,X) = nedivizibil(s Z,X) and not (Z divides X) if Z \* Z <= X .

ceq nedivizibil(Z,X) = true if Z \* Z > X .

eq prim(0) = false .

eq prim(1) = false .

ceq prim(X) = nedivizibil(2,X) if X > 1 .

endfm

fmod LISTE3 is

protecting NAT .

sort Lista .

subsort Nat < Lista .

op nil : -> Lista . \*\*\*> lista vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

\*\*\*> concatenarea listelor

\*\*\*> operatii din modulul LISTE2:

op \_apartine\_ : Nat Lista -> Bool .

op inversa : Lista -> Lista .

op stergetot : Nat Lista -> Lista .

ops elimdup elimdup2 : Lista -> Lista .

ops lungpara lungimpara : Lista -> Bool .

\*\*\*( determina daca o lista are lungimea para,

respectiv impara, fara a calcula lungimea listei )

ops simetrica sim : Lista -> Bool .

\*\*\*> fiecare determina daca o lista e simetrica

op listacifre : Nat -> Lista .

\*\*\*> lista cifrelor unui numar natural

ops palindrom pal : Nat -> Bool .

\*\*\*( fiecare determina daca un numar natural

este palindrom, adica, citit de la coada la cap,

ramane neschimbat )

op \_sublista\_ : Lista Lista -> Bool .

\*\*\*( determina daca o lista e sublista a altei

liste - cu elementele in ordine, dar nu neaparat

pe pozitii consecutive )

ops multime multime2 mult : Lista -> Bool .

\*\*\*( fiecare determina daca o lista e multime,

i. e. daca lista nu contine duplicate )

vars X Y : Nat .

vars L M : Lista .

eq X apartine nil = false .

eq X apartine (X L) = true .

ceq X apartine (Y L) = X apartine L if X =/= Y .

eq inversa(nil) = nil .

eq inversa(X L) = inversa(L) X .

eq stergetot(X,nil) = nil .

eq stergetot(X,X L) = stergetot(X,L) .

ceq stergetot(X,Y L) = Y stergetot(X,L) if X =/= Y .

eq elimdup(nil) = nil .

ceq elimdup(X L) = X elimdup(L) if not (X apartine L) .

ceq elimdup(X L) = elimdup(L) if X apartine L .

eq elimdup2(nil) = nil .

eq elimdup2(X L) = X elimdup2(stergetot(X,L)) .

eq lungpara(nil) = true .

eq lungpara(X) = false .

eq lungpara(X Y L) = lungpara(L) .

eq lungimpara(nil) = false .

eq lungimpara(X L) = lungpara(L) .

eq simetrica(L) = L == inversa(L) .

eq sim(nil) = true .

eq sim(X) = true .

eq sim(X L X) = sim(L) .

ceq sim(X L Y) = false if X =/= Y .

ceq listacifre(X) = X if X < 10 .

ceq listacifre(X) = listacifre(X quo 10) (X rem 10) if X >= 10 .

eq palindrom(X) = simetrica(listacifre(X)) .

eq pal(X) = sim(listacifre(X)) .

eq nil sublista L = true .

eq (X M) sublista nil = false .

eq (X M) sublista (X L) = M sublista L .

ceq (X M) sublista (Y L) = (X M) sublista L if X =/= Y .

eq multime(L) = L == elimdup(L) .

\*\*\*> elimdup lasa lista neschimbata ddaca lista e multime

eq multime2(L) = L == elimdup2(L) .

\*\*\*> elimdup2 lasa lista neschimbata ddaca lista e multime

eq mult(nil) = true .

eq mult(X L) = mult(L) and not (X apartine L) .

endfm

fmod PERECHI-LISTE is

extending NAT .

sorts Lista PerNat PerLista .

subsort Nat < Lista .

op nil : -> Lista . \*\*\*> lista vida

op \_\_ : Lista Lista -> Lista [assoc id: nil prec 20] .

\*\*\*> concatenarea listelor

op (\_,\_) : Nat Nat -> PerNat .

\*\*\*> operatia care construieste sortul PerNat

op {\_;\_} : Lista Lista -> PerLista [prec 30] .

\*\*\*> operatia care construieste sortul PerLista

op concat : PerLista PerLista -> PerLista .

\*\*\*> concatenarea de perechi de liste, pe componente

op perip : Lista -> PerLista .

\*\*\*( perip(L) = { L1 ; L2 }, unde:

L1 = lista elementelor din L aflate pe pozitii impare in L,

L2 = lista elementelor din L aflate pe pozitii pare in L,

cu pozitiile numerotate incepand de la 1, si dand rezultatul

printr-o singura parcurgere a listei L )

op eroare : -> PerNat .

\*\*\*( pentru tratarea separata a cazului listei vide in operatiile

minpper si minuper de mai jos; daca aceste operatii sunt aplicate

unei liste nevide, atunci recursia prin care sunt definite nu merge

pana la lista vida, ci se termina cu lista cu un singur element )

op succ : PerNat -> PerNat .

\*\*\*> operatie auxiliara pentru calcularea lui minppoz si minupoz

op minpper : PerNat PerNat -> PerNat .

\*\*\*> operatie auxiliara pentru calcularea lui minppoz

op minppoz : Lista -> PerNat .

\*\*\*( minppoz(L) = (minimul din lista L,prima pozitie a minimului in L),

calculate cu o singura parcurgere a listei L, si intoarcand eroare

atunci cand L este vida )

op minuper : PerNat PerNat -> PerNat .

\*\*\*> operatie auxiliara pentru calcularea lui minupoz

op minupoz : Lista -> PerNat .

\*\*\*( minupoz(L) = (minimul din lista L,ultima pozitie a minimului in L),

calculate cu o singura parcurgere a listei L, si intoarcand eroare

atunci cand L este vida )

vars X Y X1 Y1 : Nat .

vars L M L1 M1 : Lista .

eq concat({ L ; M },{ L1 ; M1 }) = { L L1 ; M M1 } .

eq perip(nil) = { nil ; nil } .

eq perip(X) = { X ; nil } .

eq perip(X Y L) = concat({ X ; Y },perip(L)) .

eq succ((X,Y)) = (X,s Y) .

ceq minpper((X,Y),(X1,Y1)) = (X,Y) if X <= X1 .

ceq minpper((X,Y),(X1,Y1)) = (X1,Y1) if X > X1 .

eq minppoz(nil) = eroare .

eq minppoz(X) = (X,1) .

eq minppoz(X Y L) = minpper((X,1),succ(minppoz(Y L))) .

ceq minuper((X,Y),(X1,Y1)) = (X,Y) if X < X1 .

ceq minuper((X,Y),(X1,Y1)) = (X1,Y1) if X >= X1 .

eq minupoz(nil) = eroare .

eq minupoz(X) = (X,1) .

eq minupoz(X Y L) = minuper((X,1),succ(minupoz(Y L))) .

endfm

fmod BSORT is

\*\*\*> bubblesortul pe liste cu elementele a, b si c

sort Lista .

ops a b c : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc] .

eq b a = a b .

eq c a = a c .

eq c b = b c .

endfm

fmod BUBBLESORT is

\*\*\*> bubblesortul pe liste de numere naturale

protecting NAT .

sort Lista .

subsort Nat < Lista .

op \_\_ : Lista Lista -> Lista [assoc] .

vars X Y : Nat .

ceq X Y = Y X if Y < X .

endfm

fmod TEMA6 is \*\*\*> 50 puncte

\*\*\*( Scrieti un modul pentru liste de numere naturale,

importand modulul NAT predefinit, care sa contina:

- o operatie care sa determine daca lungimea unei liste este

de forma 4n+3, cu n natural, fara a calcula lungimea listei;

- o operatie care trece un numar natural din baza 10 in baza 2,

si una care face transformarea inversa; reprezentarea unui

numar natural in baza 2 se va face sub forma unui sir de biti;

- o operatie care determina lista primilor X+1 multipli naturali

de Y, cu X si Y date ca argumente;

- o operatie care determina lista multiplilor naturali de Y mai

mici sau egali cu X, unde X si Y sunt date ca argumente;

- o operatie care determina daca o lista e permutare a multimii

primelor n numere naturale nenule, pentru un n natural;

- o operatie care determina lista tuturor pozitiilor minimului

intr-o lista, printr-o singura parcurgere a listei. )

\*\*\*( Aduceti acest modul la laboratorul de

saptamana viitoare. La fel pentru toate temele

care vor urma. Nu se vor puncta temele aduse cu

cel putin o saptamana intarziere. Se acorda punctaj

numai pentru temele aduse in decurs de o saptamana

din ziua in care au fost date. )

endfm

fmod TEMA7 is \*\*\*> 60 puncte

\*\*\*( Scrieti un modul pentru definirea numerelor rationale, care

sa foloseasca modulul pentru numere intregi din TEMA4, si sa contina

operatiile: +, - unar si binar, \*, /, precum si relatiile <=, <, >=

si >, desigur, implementate ca operatii de sort rezultat Bool.

Operatia / va servi la construirea sortului numerelor rationale. )

endfm

**LABORATOR 6**

fmod TEMA6 is

extending NAT .

sort Lista .

subsort Nat < Lista .

op nil : -> Lista . \*\*\*> lista vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

\*\*\*> operatii care au mai fost implementate:

op lungime : Lista -> Nat .

op \_apartine\_ : Nat Lista -> Bool .

op sterge : Nat Lista -> Lista .

op lung4k3 : Lista -> Bool .

\*\*\*( determina daca lungimea unei liste este

multiplu de 4 plus 3, fara a calcula efectiv

lungimea listei )

op 10to2 : Nat -> Lista .

\*\*\*> trece un numar natural din baza 10 in baza 2

op 2to10 : Lista -> Nat .

\*\*\*> trece un numar natural din baza 2 in baza 10

\*\*\*( reprezentarea unui numar natural in baza 2 se

va face sub forma unei liste nevide de cifre binare )

op multipli : Nat Nat -> Lista .

\*\*\*> lista primilor X+1 multipli naturali de Y

op mult : Nat Nat -> Lista .

\*\*\*> lista multiplilor naturali de Y mai mici sau egali cu X

op permutare : Lista -> Bool .

\*\*\*( determina daca o lista e permutare a multimii

primelor n numere naturale nenule, pentru un n natural )

op permut : Lista Nat -> Bool .

\*\*\*( determina daca o lista e permutare a multimii

primelor n numere naturale nenule, cu n natural dat )

op listpozmin : Lista -> Lista .

\*\*\*( determina lista tuturor pozitiilor minimului

intr-o lista, printr-o singura parcurgere a listei )

op aux : Lista Nat Nat Lista -> Lista .

\*\*\*> operatie auxiliara pentru scrierea lui listpozmin

\*\*\*( argumentele ei reprezinta: lista respectiva, pozitia

curenta in aceasta lista, minimul curent, lista curenta a

pozitiilor minimului )

vars X Y Z T : Nat .

vars L P : Lista .

eq lungime(nil) = 0 .

eq lungime(X L) = s lungime(L) .

eq X apartine nil = false .

eq X apartine (X L) = true .

ceq X apartine (Y L) = X apartine L if X =/= Y .

eq sterge(X,nil) = nil .

eq sterge(X,X L) = L .

ceq sterge(X,Y L) = Y sterge(X,L) if X =/= Y .

eq lung4k3(nil) = false .

eq lung4k3(X) = false .

eq lung4k3(X Y) = false .

eq lung4k3(X Y Z) = true .

eq lung4k3(X Y Z T L) = lung4k3(L) .

ceq 10to2(X) = X if X < 2 .

ceq 10to2(X) = 10to2(X quo 2) (X rem 2) if X >= 2 .

eq 2to10(X) = X .

eq 2to10(L X Y) = 2 \* 2to10(L X) + Y .

eq multipli(0,Y) = 0 .

eq multipli(s X,Y) = multipli(X,Y) (s X \* Y) .

eq mult(0,Y) = 0 .

ceq mult(s X,Y) = mult(X,Y) (s X) if Y divides (s X) .

ceq mult(s X,Y) = mult(X,Y) if not (Y divides (s X)) .

eq permutare(L) = permut(L,lungime(L)) .

eq permut(nil,0) = true .

eq permut(nil,s Y) = false .

eq permut(X L,0) = false .

eq permut(X L,s Y) = (s Y) apartine (X L) and permut(sterge(s Y,X L),Y) .

eq listpozmin(nil) = nil .

eq listpozmin(X L) = aux(X L,1,X,nil) .

eq aux(nil,Z,T,P) = P .

eq aux(X L,Z,X,P) = aux(L,s Z,X,P Z) .

ceq aux(X L,Z,T,P) = aux(L,s Z,T,P) if X > T .

ceq aux(X L,Z,T,P) = aux(L,s Z,X,Z) if X < T .

endfm

fmod TEMA7 is

sorts Zero NzNat Nat Neg NzInt Int NzRat Rat .

subsort Zero < Nat .

subsorts NzNat < Nat NzInt < Int < Rat .

subsorts Neg < NzInt < NzRat .

op 0 : -> Zero .

ops s\_ p\_ : Int -> Int .

op s\_ : Nat -> NzNat .

op p\_ : NzNat -> Nat .

op p\_ : Zero -> Neg .

op p\_ : Neg -> Neg .

ops (\_+\_) (\_\*\_) (\_-\_) : Int Int -> Int .

ops (\_+\_) (\_\*\_) : Nat Nat -> Nat .

op \_+\_ : NzNat Nat -> NzNat .

op \_+\_ : Nat NzNat -> NzNat .

op \_\*\_ : NzInt NzInt -> NzInt .

op \_\*\_ : NzNat NzNat -> NzNat .

op \_\*\_ : Zero Int -> Zero .

op \_\*\_ : Int Zero -> Zero .

op -\_ : Int -> Int .

op -\_ : NzNat -> Neg .

op -\_ : Neg -> NzNat .

op \_/\_ : Int NzInt -> Rat .

op \_/\_ : NzInt NzInt -> NzRat .

op \_/\_ : Rat NzRat -> Rat .

op \_/\_ : NzRat NzRat -> NzRat .

ops (\_+\_) (\_\*\_) (\_-\_) : Rat Rat -> Rat .

op \_\*\_ : NzRat NzRat -> NzRat .

op -\_ : Rat -> Rat .

op -\_ : NzRat -> NzRat .

op abs : Int -> Nat .

op abs : NzInt -> NzNat .

ops (\_<=\_) (\_<\_) (\_>=\_) (\_>\_) : Rat Rat -> Bool .

op \_div\_ : Nat NzNat -> Nat .

op \_div\_ : Int NzInt -> Int .

op \_mod\_ : Int NzInt -> Nat .

op cmmdc : Int Int -> Nat .

vars X Y : Int .

vars Z T U : NzInt .

vars M M1 : Nat .

vars N N1 : Neg .

vars P S : NzNat .

vars Q R : Rat .

eq p s X = X .

eq s p X = X .

eq X + 0 = X .

eq X + s Y = s (X + Y) .

eq X + p Y = p (X + Y) .

eq - 0 = 0 .

eq - - X = X .

eq - s X = p (- X) .

eq - p X = s (- X) .

eq X - 0 = X .

eq X - s Y = p (X - Y) .

eq X - p Y = s (X - Y) .

eq X \* 0 = 0 .

eq X \* s Y = (X \* Y) + X .

eq X \* p Y = (X \* Y) - X .

eq abs(M) = M .

eq abs(N) = - N .

eq 0 <= M = true .

eq P <= 0 = false .

eq - M <= - M1 = M1 <= M .

eq N <= M = true .

eq M <= N = false .

eq s X <= s Y = X <= Y .

eq p X <= p Y = X <= Y .

eq Q < R = Q <= R and Q =/= R .

eq Q >= R = R <= Q .

eq Q > R = R < Q .

ceq M div P = 0 if M < P .

ceq M div P = s((M - P) div P) if P <= M .

ceq M mod P = M if M < P .

ceq M mod P = (M - P) mod P if P <= M .

eq cmmdc(0,M) = M .

eq cmmdc(P,M) = cmmdc(M mod P,P) .

ceq cmmdc(X,Y) = cmmdc(abs(X),abs(Y)) if X < 0 or Y < 0 .

eq X mod N = X mod (- N) .

ceq N mod P = 0 if (- N) mod P == 0 .

ceq N mod P = P - ((- N) mod P) if (- N) mod P =/= 0 .

eq X div N = - (X div (- N)) .

ceq N div P = - ((- N) div P) if (- N) mod P == 0 .

ceq N div P = p (- ((- N) div P)) if (- N) mod P =/= 0 .

eq X / (s 0) = X .

ceq X / Z = (X div cmmdc(X,Z)) / (Z div cmmdc(X,Z)) if cmmdc(X,Z) =/= s 0 .

eq (X / Z) / (T / U) = (X \* U) / (Z \* T) .

eq X / (T / U) = (X \* U) / T .

eq (X / Z) / T = X / (Z \* T) .

eq (X / Z) \* (Y / T) = (X \* Y) / (Z \* T) .

eq X \* (Y / T) = (X \* Y) / T .

eq (X / Z) \* Y = (X \* Y) / Z .

eq (X / Z) + (Y / T) = ((X \* T) + (Y \* Z)) / (Z \* T) .

eq X + (Y / T) = ((X \* T) + Y) / T .

eq (X / Z) + Y = (X + (Y \* Z)) / Z .

eq (X / Z) - (Y / T) = ((X \* T) - (Y \* Z)) / (Z \* T) .

eq X - (Y / T) = ((X \* T) - Y) / T .

eq (X / Z) - Y = (X - (Y \* Z)) / Z .

eq - (X / Z) = (- X) / Z .

eq X / P <= Y / S = X \* S <= Y \* P .

eq X / N <= Y / N1 = Y / (- N1) <= X / (- N) .

eq X / N <= Y / S = (- X) / (- N) <= Y / S .

eq X / P <= Y / N1 = X / P <= (- Y) / (- N1) .

eq X <= Y / S = X \* S <= Y .

eq X <= Y / N1 = X <= (- Y) / (- N1) .

eq X / Z <= Y = (- Y) <= (- X) / Z .

endfm

fmod MULTIMI is

\*\*\*> operatii cu multimi

\*\*\*> multimile vor fi reprezentate ca liste fara duplicate

protecting NAT .

sort Multime .

subsort Nat < Multime .

op nil : -> Multime .

op \_\_ : Multime Multime -> Multime [assoc comm id: nil prec 20] .

\*\*\*> Urmatoarele doua operatii au mai fost implementate:

op \_apartine\_ : Nat Multime -> Bool [prec 30] .

op elimdup : Multime -> Multime .

\*\*\*> transforma o lista intr-o multime, prin eliminarea duplicatelor

\*\*\*( Privind operatiile cu multimi de mai jos, presupunem ca acestea

vor fi intotdeauna aplicate unor multimi, i. e. niciodata nu vor primi

ca argumente liste cu duplicate. Ca sa le putem aplica si listelor cu

duplicate, am avea nevoie de operatii auxiliare; de exemplu, pentru

intersectie, am avea nevoie de o operatie auxiliara de intersectie:

op \_I2\_ : Multime Multime -> Multime [assoc comm prec 30] .

definita astfel:

eq L I2 M = elimdup(L) I elimdup(M) .

sau astfel:

eq L I2 M = elimdup(L I M) .

Singura operatie de mai jos care nu ar necesita acest artificiu este

operatia \_V\_ (reuniunea calculata cu a doua metoda). )

ops (\_U\_) (\_V\_) : Multime Multime -> Multime [assoc comm prec 30] .

\*\*\*> reuniunea, calculata prin doua metode

op \_I\_ : Multime Multime -> Multime [assoc comm prec 30] .

\*\*\*> intersectia

op \_\\_ : Multime Multime -> Multime [prec 30] .

\*\*\*> diferenta

ops (\_D\_) (\_-\_) : Multime Multime -> Multime [assoc comm prec 30] .

\*\*\*> diferenta simetrica

vars X Y : Nat .

vars L M : Multime .

eq X apartine nil = false .

eq X apartine X M = true .

ceq X apartine Y M = X apartine M if X =/= Y .

eq elimdup(nil) = nil .

ceq elimdup(X M) = X elimdup(M) if not (X apartine M) .

ceq elimdup(X M) = elimdup(M) if X apartine M .

eq nil U M = M .

ceq X L U M = X (L U M) if not (X apartine M) .

ceq X L U M = L U M if X apartine M .

eq L V M = elimdup(L M) .

eq nil I M = nil .

ceq X L I M = L I M if not (X apartine M) .

ceq X L I M = X (L I M) if X apartine M .

eq nil \ M = nil .

ceq X L \ M = X (L \ M) if not (X apartine M) .

ceq X L \ M = L \ M if X apartine M .

eq L D M = (L \ M) U (M \ L) .

eq L - M = (L \ M) V (M \ L) .

endfm

fmod PERMCIRC is

\*\*\*( generarea permutarilor circulare ale unei

liste cu sau fara duplicate )

\*\*\*> Acest modul exemplifica lucrul cu liste de liste.

\*\*\*> De fapt nu avem nevoie aici de listele vide.

protecting NAT .

sorts Lista ListLista .

subsorts Nat < Lista < ListLista .

\*\*\*> Lista = sortul pentru liste de numere naturale

\*\*\*> ListLista = sortul pentru liste de liste de numere naturale

op nil : -> Lista .

\*\*\*( lista vida de numere naturale: elementul neutru la

concatenarea de liste de numere naturale )

op \_\_ : Lista Lista -> Lista [assoc id: nil prec 20] .

\*\*\*> concatenarea de liste de numere naturale

op null : -> ListLista .

\*\*\*( lista vida de liste de numere naturale: elementul neutru la

concatenarea de liste de liste de numere naturale )

op \_;\_ : ListLista ListLista -> ListLista [assoc id: null prec 30] .

\*\*\*> concatenarea de liste de liste de numere naturale

op genperm : Lista -> ListLista .

\*\*\*> operatia care genereaza permutarile circulare

op permuta : Lista Lista -> ListLista .

\*\*\*> operatie auxiliara pentru scrierea lui genperm

var X : Nat .

vars L M : Lista .

eq genperm(nil) = nil . \*\*\*( lista de liste de numere

naturale cu unicul element dat de lista vida de numere

naturale (lista de numere naturale fara elemente) )

eq genperm(L X) = permuta(L X,X L) .

eq permuta(L,L) = L .

ceq permuta(L,M X) = M X ; permuta(L,X M) if L =/= M X .

endfm

fmod PERMCIRC2 is

\*\*\*( Generarea permutarilor circulare ale unei

liste cu sau fara duplicate, versiunea 2: fara

liste vide. )

protecting NAT .

sorts Lista ListLista .

subsorts Nat < Lista < ListLista .

\*\*\*> Lista = sortul pentru liste de numere naturale

\*\*\*> ListLista = sortul pentru liste de liste de numere naturale

op \_\_ : Lista Lista -> Lista [assoc prec 20] .

\*\*\*> concatenarea de liste de numere naturale

op \_;\_ : ListLista ListLista -> ListLista [assoc prec 30] .

\*\*\*> concatenarea de liste de liste de numere naturale

op genperm : Lista -> ListLista .

\*\*\*> operatia care genereaza permutarile circulare

op permuta : Lista Lista -> ListLista .

\*\*\*> operatie auxiliara pentru scrierea lui genperm

var X : Nat .

vars L M : Lista .

eq genperm(X) = X .

eq genperm(L X) = permuta(L X,X L) .

eq permuta(L,L) = L .

ceq permuta(L,M X) = M X ; permuta(L,X M) if L =/= M X .

endfm

fmod LISTENAT is

\*\*\*> liste de numere naturale, cu lista vida si concatenarea

\*\*\*> modul pe care il vom importa in modulele urmatoare, pentru sortari

protecting NAT .

sort Lista .

subsort Nat < Lista .

op nil : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

endfm

fmod INSSORT is

\*\*\*> sortarea prin insertie directa

extending LISTENAT .

op sortare : Lista -> Lista .

op inserare : Nat Lista -> Lista .

vars X Y : Nat .

var L : Lista .

eq sortare(nil) = nil .

eq sortare(X L) = inserare(X,sortare(L)) .

eq inserare(X,nil) = X .

ceq inserare(X,Y L) = Y inserare(X,L) if Y < X .

ceq inserare(X,Y L) = X Y L if X <= Y .

endfm

fmod MERGESORT is

\*\*\*> sortarea prin interclasare

\*\*\*( Intrucat oricum taierea unei liste in doua parti,

indiferent de metoda de taiere, se face in timp liniar,

nu constant, ca intr-o implementare a acestui algoritm

de sortare intr-un limbaj de programare imperativa, putem

scrie

acest algoritm in Maude, pur si simplu, ca mai jos. )

extending LISTENAT .

op interclasare : Lista Lista -> Lista .

\*\*\*> interclaseaza doua liste sortate

op sortare : Lista -> Lista .

vars X Y : Nat .

vars L M : Lista .

eq interclasare(L,nil) = L .

eq interclasare(nil,M) = M .

ceq interclasare(X L,Y M) = X interclasare(L,Y M) if X <= Y .

ceq interclasare(X L,Y M) = Y interclasare(X L,M) if X > Y .

eq sortare(nil) = nil .

eq sortare(X) = X .

eq sortare(X L Y M) = interclasare(sortare(X L),sortare(Y M)) .

endfm

fmod SELSORT is

\*\*\*> sortarea prin selectia directa a minimului

extending LISTENAT .

op minim : Nat Nat -> Nat .

op minlist : Lista -> Nat .

op sterge : Nat Lista -> Lista .

op sortare : Lista -> Lista .

vars X Y : Nat .

var L : Lista .

ceq minim(X,Y) = X if X <= Y .

ceq minim(X,Y) = Y if X > Y .

eq minlist(X) = X .

eq minlist(X Y L) = min(X,minlist(Y L)) .

eq sterge(X,nil) = nil .

eq sterge(X,X L) = L .

ceq sterge(X,Y L) = Y sterge(X,L) if X =/= Y .

eq sortare(nil) = nil .

eq sortare(X L) = minlist(X L) sortare(sterge(minlist(X L),X L)) .

endfm

fmod TEMA8 is \*\*\*> 20 puncte

\*\*\*> sortarea prin selectia directa a maximului

\*\*\*( Aduceti temele din aceasta lectie in saptamana

urmatoare celei in care dam lucrarea de control. )

endfm

fmod TEMA9 is \*\*\*> 50 puncte

\*\*\*> bubblesortul, efectuat intr-o operatie distincta de concatenare

\*\*\*( Dupa cum probabil ati observat, sortarea in operatia de

concatenare este facuta automat de Maude, daca dam atributul comm

in declaratia operatiei de concatenare. In anumite probleme insa,

putem avea nevoie si de liste nesortate, si de sortarea acestor liste.

In astfel de situatii, desigur, va conta pozitia elementelor intr-o

Lista, asadar concatenarea nu poate fi declarata comutativa, iar

Sortarea va trebui sa fie efectuata intr-o operatie distincta de

operatia de concatenare. )

\*\*\*( Sa presupunem ca, intr-o problema de tipul descris mai sus, dorim

sa efectuam sortarea prin metoda bulelor. Aceasta este problema din

tema 9. )

endfm

fmod TEMA10 is \*\*\*> 150 puncte

\*\*\*> generarea tuturor permutarilor unei liste cu sau fara duplicate

\*\*\*> (nu numai a celor circulare)

endfm

fmod TEMA11 is \*\*\*> 100 puncte

\*\*\*> problema celor N dame

\*\*\*( Pentru N natural nenul, sa se aseze N dame pe o tabla de sah NxN,

astfel incat sa nu se atace una pe alta. Sa se dea toate solutiile

posibile. Nu exista solutie pentru orice N natural nenul. )

\*\*\*( Solutia va fi data sub forma de lista de liste de numere naturale,

reprezentand lista tuturor configuratiilor posibile, unde o configuratie

va fi lista coloanelor pe care se afla damele de pe liniile 1,2,...,N,

respectiv. Desigur, aceste coloane vor fi doua cate doua distincte, pentru

ca damele sa nu se atace pe coloana. )

endfm

**LABORATOR 7\_8**

fmod QUICKSORT is

\*\*\*> sortarea rapida

\*\*\*( OBSERVATIE: calculele de complexitate pentru algoritmi

din programarea imperativa nu sunt valabile aici. )

protecting NAT .

sorts Lista PerecheListe .

subsort Nat < Lista .

op nil : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc id: nil prec 20] .

op {\_;\_} : Lista Lista -> PerecheListe [prec 30] .

\*\*\*> operatia care construieste sortul PerecheListe

op concat : PerecheListe PerecheListe -> PerecheListe .

\*\*\*> concatenarea pe componente

op per-lista : PerecheListe -> Lista .

op taie : Nat Lista -> PerecheListe .

op qsort : Lista -> Lista .

op qsortper : PerecheListe -> PerecheListe .

vars X Y : Nat .

vars L L1 T T1 : Lista .

eq concat({ L ; L1 },{ T ; T1 }) = { L T ; L1 T1 } .

\*\*\*> in cele ce urmeaza, X va fi pivotul

eq taie(X,nil) = { nil ; nil } .

ceq taie(X,Y L) = concat({ Y ; nil },taie(X,L)) if Y <= X .

ceq taie(X,Y L) = concat({ nil ; Y },taie(X,L)) if Y > X .

eq per-lista({ L ; L1 }) = L L1 .

eq qsort(nil) = nil .

eq qsort(X L) = per-lista(concat(qsortper(taie(X,L)),{ X ; nil })) .

\*\*\*( este esential sa nu includem pivotul in lista care trebuie taiata,

pentru ca avem nevoie sa micsoram lungimea listei )

eq qsortper({ L ; L1 }) = { qsort(L) ; qsort(L1) } .

endfm

fmod OBSERVATII is

\*\*\*( Operatorul if\_then\_else\_fi si atributul owise (= "otherwise"):

owise este un atribut de ecuatie, nu unul de operatie, ca assoc, comm,

id:, prec etc.: )

\*\*\*( In modulul urmator, este incomod de folosit operatia listnrap

pentru implementarea operatiei nrapct, dar ar fi util daca, in ultimele

doua ecuatii, am evita recalcularea lui nrap(X,L) si nrap(Y,L) (care se

efectueaza daca prima dintre acele doua ecuatii pe care Maude-ul incearca

sa o aplice nu este cea aplicabila in cazul curent). )

\*\*\*( Solutia 1: utilizarea operatorului if\_then\_else\_fi, si scrierea

acelor doua ecuatii in una singura, care va fi o ecuatie NECONDITIONATA:

eq auxiliara(L,X R,C Y) = if nrap(X,L) == nrap(Y,L) then auxiliara(L,R,C Y X) else C Y ; auxiliara(L,R,X) fi .

)

\*\*\*( Solutia 2: scrierea tot cu doua ecuatii, dintre care prima conditionata,

iar a doua neconditionata si continand atributul owise, care ne permite sa

nu mai scriem in aceasta a doua ecuatie negatia conditiei din prima ecuatie:

ceq auxiliara(L,X R,C Y) = auxiliara(L,R,C Y X) if nrap(X,L) == nrap(Y,L) .

eq auxiliara(L,X R,C Y) = C Y ; auxiliara(L,R,X) [owise] .

)

endfm

fmod SUBIECTUL4 is

extending NAT .

sorts Lista ListLista .

subsorts Nat < Lista < ListLista .

op nil : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc id: nil prec 20] .

op null : -> ListLista .

op \_;\_ : ListLista ListLista -> ListLista [assoc id: null prec 30] .

op nrparap : Lista -> Bool .

\*\*\*(

nrparap(L) = true, daca toate elementele listei L au numar par

de aparitii in lista L, si

nrparap(L) = false, in caz contrar.

)

op maxmaxap : Lista -> Bool .

\*\*\*(

maxmaxap(L) = true daca maximul din lista L are cele mai multe

aparitii in lista L, comparativ cu celelalte elemente ale listei L,

si

maxmaxap(L) = false, in caz contrar.

)

op nrapct : Lista -> ListLista .

\*\*\*(

nrapct(L) = lista sublistelor S ale lui L care satisfac urmatoarele

conditii: elementele lui S se afla pe pozitii consecutive in lista L

si au acelasi numar de aparitii in lista L, si sublista S nu poate fi

prelungita cu niciun element din lista L astfel incat sa se pastreze

proprietatile anterioare.

)

\*\*\*> Operatii auxiliare necesare (sau comod de folosit):

op maxlist : Lista -> Nat .

\*\*\*( determina maximul dintr-o lista nevida, folosind

operatia max din modulul predefinit NAT, operatie care

determina maximul a doua numere naturale )

op nrap : Nat Lista -> Nat .

\*\*\*> numara aparitiile unui element intr-o lista

op listnrap : Lista -> Lista .

\*\*\*( inlocuieste, intr-o lista, fiecare element cu numarul aparitiilor

sale in lista initiala, folosind operatia auxiliara aux, care retine si

lista initiala, si restul de lista care mai trebuie parcurs )

op aux : Lista Lista -> Lista .

op toatepare : Lista -> Bool .

\*\*\*> testeaza daca o lista are toate elementele pare

op auxiliara : Lista Lista Lista -> ListLista .

\*\*\*> operatie auxiliara pentru scrierea lui nrapct

\*\*\*( retine lista initiala, restul de lista care mai trebuie parcurs,

si sublista curenta )

op decide : ListLista -> ListLista .

\*\*\*> alta operatie auxiliara pentru scrierea lui nrapct

\*\*\*> a se vedea mai jos definitia ei

vars X Y : Nat .

vars L R C : Lista .

var LL : ListLista .

eq maxlist(nil) = 0 .

eq maxlist(X L) = max(X,maxlist(L)) .

eq nrap(X,nil) = 0 .

eq nrap(X,X L) = s nrap(X,L) .

ceq nrap(X,Y L) = nrap(X,L) if X =/= Y .

eq listnrap(L) = aux(L,L) .

eq aux(L,nil) = nil .

eq aux(L,X C) = nrap(X,L) aux(L,C) .

eq toatepare(nil) = true .

eq toatepare(X L) = 2 divides X and toatepare(L) .

eq nrparap(L) = toatepare(listnrap(L)) .

eq maxmaxap(L) = nrap(maxlist(L),L) == maxlist(listnrap(L)) .

eq nrapct(L) = decide(auxiliara(L,L,nil)) .

eq decide(null) = nil .

eq decide(L ; LL) = L ; LL .

eq auxiliara(L,nil,nil) = null .

eq auxiliara(L,nil,C Y) = C Y .

eq auxiliara(L,X R,nil) = auxiliara(L,R,X) .

ceq auxiliara(L,X R,C Y) = auxiliara(L,R,C Y X) if nrap(X,L) == nrap(Y,L) .

ceq auxiliara(L,X R,C Y) = C Y ; auxiliara(L,R,X) if nrap(X,L) =/= nrap(Y,L) .

endfm

fmod SOLUTIA1 is

extending NAT .

sorts Lista ListLista .

subsorts Nat < Lista < ListLista .

op nil : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc id: nil prec 20] .

op null : -> ListLista .

op \_;\_ : ListLista ListLista -> ListLista [assoc id: null prec 30] .

op nrparap : Lista -> Bool .

\*\*\*(

nrparap(L) = true, daca toate elementele listei L au numar par

de aparitii in lista L, si

nrparap(L) = false, in caz contrar.

)

op maxmaxap : Lista -> Bool .

\*\*\*(

maxmaxap(L) = true daca maximul din lista L are cele mai multe

aparitii in lista L, comparativ cu celelalte elemente ale listei L,

si

maxmaxap(L) = false, in caz contrar.

)

op nrapct : Lista -> ListLista .

\*\*\*(

nrapct(L) = lista sublistelor S ale lui L care satisfac urmatoarele

conditii: elementele lui S se afla pe pozitii consecutive in lista L

si au acelasi numar de aparitii in lista L, si sublista S nu poate fi

prelungita cu niciun element din lista L astfel incat sa se pastreze

proprietatile anterioare.

)

\*\*\*> Operatii auxiliare necesare (sau comod de folosit):

op maxlist : Lista -> Nat .

\*\*\*( determina maximul dintr-o lista nevida, folosind

operatia max din modulul predefinit NAT, operatie care

determina maximul a doua numere naturale )

op nrap : Nat Lista -> Nat .

\*\*\*> numara aparitiile unui element intr-o lista

op listnrap : Lista -> Lista .

\*\*\*( inlocuieste, intr-o lista, fiecare element cu numarul aparitiilor

sale in lista initiala, folosind operatia auxiliara aux, care retine si

lista initiala, si restul de lista care mai trebuie parcurs )

op aux : Lista Lista -> Lista .

op toatepare : Lista -> Bool .

\*\*\*> testeaza daca o lista are toate elementele pare

op auxiliara : Lista Lista Lista -> ListLista .

\*\*\*> operatie auxiliara pentru scrierea lui nrapct

\*\*\*( retine lista initiala, restul de lista care mai trebuie parcurs,

si sublista curenta )

op decide : ListLista -> ListLista .

\*\*\*> alta operatie auxiliara pentru scrierea lui nrapct

\*\*\*> a se vedea mai jos definitia ei

vars X Y : Nat .

vars L R C : Lista .

var LL : ListLista .

eq maxlist(nil) = 0 .

eq maxlist(X L) = max(X,maxlist(L)) .

eq nrap(X,nil) = 0 .

eq nrap(X,X L) = s nrap(X,L) .

ceq nrap(X,Y L) = nrap(X,L) if X =/= Y .

eq listnrap(L) = aux(L,L) .

eq aux(L,nil) = nil .

eq aux(L,X C) = nrap(X,L) aux(L,C) .

eq toatepare(nil) = true .

eq toatepare(X L) = 2 divides X and toatepare(L) .

eq nrparap(L) = toatepare(listnrap(L)) .

eq maxmaxap(L) = nrap(maxlist(L),L) == maxlist(listnrap(L)) .

eq nrapct(L) = decide(auxiliara(L,L,nil)) .

eq decide(null) = nil .

eq decide(L ; LL) = L ; LL .

eq auxiliara(L,nil,nil) = null .

eq auxiliara(L,nil,C Y) = C Y .

eq auxiliara(L,X R,nil) = auxiliara(L,R,X) .

eq auxiliara(L,X R,C Y) = if nrap(X,L) == nrap(Y,L) then auxiliara(L,R,C Y X) else C Y ; auxiliara(L,R,X) fi .

endfm

fmod SOLUTIA2 is

extending NAT .

sorts Lista ListLista .

subsorts Nat < Lista < ListLista .

op nil : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc id: nil prec 20] .

op null : -> ListLista .

op \_;\_ : ListLista ListLista -> ListLista [assoc id: null prec 30] .

op nrparap : Lista -> Bool .

\*\*\*(

nrparap(L) = true, daca toate elementele listei L au numar par

de aparitii in lista L, si

nrparap(L) = false, in caz contrar.

)

op maxmaxap : Lista -> Bool .

\*\*\*(

maxmaxap(L) = true daca maximul din lista L are cele mai multe

aparitii in lista L, comparativ cu celelalte elemente ale listei L,

si

maxmaxap(L) = false, in caz contrar.

)

op nrapct : Lista -> ListLista .

\*\*\*(

nrapct(L) = lista sublistelor S ale lui L care satisfac urmatoarele

conditii: elementele lui S se afla pe pozitii consecutive in lista L

si au acelasi numar de aparitii in lista L, si sublista S nu poate fi

prelungita cu niciun element din lista L astfel incat sa se pastreze

proprietatile anterioare.

)

\*\*\*> Operatii auxiliare necesare (sau comod de folosit):

op maxlist : Lista -> Nat .

\*\*\*( determina maximul dintr-o lista nevida, folosind

operatia max din modulul predefinit NAT, operatie care

determina maximul a doua numere naturale )

op nrap : Nat Lista -> Nat .

\*\*\*> numara aparitiile unui element intr-o lista

op listnrap : Lista -> Lista .

\*\*\*( inlocuieste, intr-o lista, fiecare element cu numarul aparitiilor

sale in lista initiala, folosind operatia auxiliara aux, care retine si

lista initiala, si restul de lista care mai trebuie parcurs )

op aux : Lista Lista -> Lista .

op toatepare : Lista -> Bool .

\*\*\*> testeaza daca o lista are toate elementele pare

op auxiliara : Lista Lista Lista -> ListLista .

\*\*\*> operatie auxiliara pentru scrierea lui nrapct

\*\*\*( retine lista initiala, restul de lista care mai trebuie parcurs,

si sublista curenta )

op decide : ListLista -> ListLista .

\*\*\*> alta operatie auxiliara pentru scrierea lui nrapct

\*\*\*> a se vedea mai jos definitia ei

vars X Y : Nat .

vars L R C : Lista .

var LL : ListLista .

eq maxlist(nil) = 0 .

eq maxlist(X L) = max(X,maxlist(L)) .

eq nrap(X,nil) = 0 .

eq nrap(X,X L) = s nrap(X,L) .

ceq nrap(X,Y L) = nrap(X,L) if X =/= Y .

eq listnrap(L) = aux(L,L) .

eq aux(L,nil) = nil .

eq aux(L,X C) = nrap(X,L) aux(L,C) .

eq toatepare(nil) = true .

eq toatepare(X L) = 2 divides X and toatepare(L) .

eq nrparap(L) = toatepare(listnrap(L)) .

eq maxmaxap(L) = nrap(maxlist(L),L) == maxlist(listnrap(L)) .

eq nrapct(L) = decide(auxiliara(L,L,nil)) .

eq decide(null) = nil .

eq decide(L ; LL) = L ; LL .

eq auxiliara(L,nil,nil) = null .

eq auxiliara(L,nil,C Y) = C Y .

eq auxiliara(L,X R,nil) = auxiliara(L,R,X) .

ceq auxiliara(L,X R,C Y) = auxiliara(L,R,C Y X) if nrap(X,L) == nrap(Y,L) .

eq auxiliara(L,X R,C Y) = C Y ; auxiliara(L,R,X) [owise] .

endfm

fmod DIN-TEMA is

\*\*\*( Operatii cu siruri de biti: and, or, xor pe biti,

si adunarea in binar, efectuata direct pe sirurile de biti. )

\*\*\*( Adunarea, scaderea si inmultirea polinoamelor cu

coeficienti intregi, peste INT-ul predefinit. )

endfm

**LABORATOR 9**

fmod ARBBIN is

\*\*\*> parcurgerile arborilor binari

protecting NAT .

sorts Lista Arbbin .

subsort Nat < Lista .

\*\*\*> Lista e sortul pentru liste de numere naturale

\*\*\*> Arbbin e sortul pentru arbori binari

op nil : -> Lista . \*\*\*> lista de numere naturale vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbbin . \*\*\*> arborele binar vid

op \_{\_,\_} : Nat Arbbin Arbbin -> Arbbin .

\*\*\*> operatiile care construiesc sortul Arbbin

\*\*\*( arborii binari nevizi sunt arborii binari care au radacina,

adica arborii binari de forma X{A,B}, cu X de sort Nat, reprezentand

informatia din radacina, si A si B de sort Arbbin, reprezentand

subarborele stang si, respectiv, subarborele drept )

ops pre ino post : Arbbin -> Lista .

\*\*\*> parcurgerile in preordine, inordine, respectiv postordine

ops ex1 ex2 : -> Arbbin \*\*\*> exemple de arbori binari (vezi mai jos)

var X : Nat .

vars A B : Arbbin .

eq pre(null) = nil .

eq pre(X{A,B}) = X pre(A) pre(B) .

eq ino(null) = nil .

eq ino(X{A,B}) = ino(A) X ino(B) .

eq post(null) = nil .

eq post(X{A,B}) = post(A) post(B) X .

eq ex1 = 1{2{null,null},3{null,4{null,null}}} .

eq ex2 = 1{2{4{null,null},5{6{null,null},null}},3{null,7{8{null,null},9{10{null,null},null}}}} .

\*\*\*( putem da: red pre(ex1) . red ino(ex1) . red post(ex1) .

red pre(ex2) . red ino(ex2) . red post(ex2) . )

endfm

fmod ARBBIN-LF is

\*\*\*> exercitiu cu arbori binari: lista frunzelor

protecting ARBBIN .

op lf : Arbbin -> Lista .

\*\*\*> lista frunzelor unui arbore binar

\*\*\*( Frunzele sunt arborii (sau subarborii) binari de forma

X{null,null}, cu X de sort Nat. )

var X : Nat .

vars A B : Arbbin .

eq lf(null) = nil .

eq lf(X{null,null}) = X .

ceq lf(X{A,B}) = lf(A) lf(B) if A =/= null or B =/= null .

\*\*\*> putem da: red lf(ex1) . red lf(ex2) .

endfm

fmod ARBBINCAUT is

\*\*\*> arbori binari de cautare

\*\*\*( Arborii binari de cautare sunt arborii binari cu proprietatea

ca informatia din fiecare nod N este mai mare sau egala cu informatia

din fiecare nod din subarborele stang al lui N si mai mica decat

informatia din fiecare nod din subarborele drept al lui N (de fapt,

egalitatea poate fi considerata pentru oricare dintre subarbori, iar

inegalitatea stricta pentru celalalt subarbore, dar aceasta regula

trebuie sa fie respectata pentru toate nodurile arborelui). Parcurgand

in inordine un arbore binar de cautare, obtinem lista sortata a

valorilor din nodurile acelui arbore. )

\*\*\*( Vom face sortare cu arbori binari de cautare. Am putea retine

in nodurile unui astfel de arbore si numarul nodului si informatia

din noduri, dar vom retine numai informatia, adica valorile din

lista pe care o vom sorta. )

extending ARBBIN .

op insert : Nat Arbbin -> Arbbin .

\*\*\*( insereaza un numar natural intr-un arbore binar de cautare,

astfel incat arborele obtinut prin inserare sa fie tot un arbore

binar de cautare )

op insertlist : Lista Arbbin -> Arbbin .

\*\*\*( insereaza o lista de numere naturale intr-un arbore binar de

cautare, astfel incat arborele obtinut prin inserare sa fie tot

un arbore binar de cautare )

op sortare : Lista -> Lista .

\*\*\*( sorteaza o lista de numere naturale, folosind

un arbore binar de cautare )

vars X Y : Nat .

var L : Lista .

vars A B : Arbbin .

eq insert(Y,null) = Y{null,null} .

eq insert(Y,X{A,B}) = if Y <= X then X{insert(Y,A),B} else X{A,insert(Y,B)} fi .

eq insertlist(nil,A) = A .

eq insertlist(X L,A) = insert(X,insertlist(L,A)) .

eq sortare(L) = ino(insertlist(L,null)) .

\*\*\*( putem vedea si arborele de cautare creat pentru sortarea

unei liste L, cu: red insertlist(L,null) . )

endfm

fmod ARBORE is

\*\*\*> arbori oarecare

protecting NAT .

sorts Lista Arbore ListArb .

subsort Nat < Lista .

subsort Arbore < ListArb .

\*\*\*> Lista e sortul pentru liste de numere naturale

\*\*\*> Arbore e sortul pentru arbori oarecare

\*\*\*> ListArb e sortul pentru liste de arbori oarecare

op nil : -> Lista . \*\*\*> lista de numere naturale vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbore . \*\*\*> arborele vid

op \_{\_} : Nat ListArb -> Arbore [prec 20] .

\*\*\*> operatiile care construiesc sortul Arbore

\*\*\*( arborii nevizi sunt arborii care au radacina, adica arborii de

forma X{LA}, cu X informatia din radacina si lista subarborilor LA )

op frunza : -> ListArb . \*\*\*> lista de arbori vida

\*\*\*> nu este acelasi lucru cu arborele vid, null

\*\*\*( null, fiind arbore, este lista de arbori, datorita ordonarii pe

sorturi, dar este lista de arbori cu 1 element, nu lista vida de arbori )

op \_;\_ : ListArb ListArb -> ListArb [assoc id: frunza prec 30] .

\*\*\*> concatenarea de liste de arbori

op ex : -> Arbore . \*\*\*> un exemplu de arbore oarecare

eq ex = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza}} ; 7{frunza}} .

endfm

fmod ARBORE-LF is

\*\*\*> un exercitiu cu arbori oarecare: lista frunzelor

protecting ARBORE .

op lf : Arbore -> Lista .

\*\*\*> lista frunzelor unui arbore oarecare

op lflist : ListArb -> Lista .

\*\*\*> lista frunzelor unei liste de arbori oarecare

\*\*\*( Frunzele sunt arborii (sau subarborii) oarecare

de forma X{frunza}, cu X de sort Nat. )

var X : Nat .

var A : Arbore .

var LA : ListArb .

eq lf(null) = nil .

eq lf(X{frunza}) = X .

eq lf(X{A ; LA}) = lf(A) lflist(LA) .

eq lflist(frunza) = nil .

eq lflist(A ; LA) = lf(A) lflist(LA) .

\*\*\*> putem da: red lf(ex) .

endfm

fmod ARBORE-DF is

\*\*\*> parcurgerea in adancime a arborilor oarecare

protecting ARBORE .

op df : Arbore -> Lista .

\*\*\*> parcurge in adancime un arbore oarecare

op dflist : ListArb -> Lista .

\*\*\*> parcurge in adancime o lista de arbori oarecare

var X : Nat .

var A : Arbore .

var LA : ListArb .

eq df(null) = nil .

eq df(X{LA}) = X dflist(LA) .

eq dflist(frunza) = nil .

eq dflist(A ; LA) = df(A) dflist(LA) .

\*\*\*> putem da: red df(ex) .

endfm

fmod ARBORE-BF is

\*\*\*> parcurgerea arborilor oarecare in latime, adica pe niveluri

protecting ARBORE .

op bf : Arbore -> Lista .

\*\*\*> parcurge in latime un arbore oarecare

op bflist : ListArb -> Lista .

\*\*\*> parcurge in latime o lista de arbori oarecare

var X : Nat .

var L : Lista .

var A : Arbore .

vars LA LA1 : ListArb .

eq bf(null) = nil .

eq bf(X{LA}) = X bflist(LA) .

eq bflist(frunza) = nil .

eq bflist(X{LA} ; LA1) = X bflist(LA1 ; LA) .

\*\*\*( Dupa cum se vede, am folosit o coada, sau,

mai exact, am simulat folosirea unei cozi. )

\*\*\*> putem da: red bf(ex) .

endfm

fmod OBSERVATIE-TEME is

\*\*\*( Aceste teme trebuie aduse saptamana viitoare, odata cu cele date in

lectia anterioara, enuntate la sfarsitul fisierului modlectia7\_8.maude. )

endfm

fmod TEMA15 is \*\*\*> 100 puncte

\*\*\*( Operatii cu arbori binari:

- parcurgerea unui arbore binar in ordinea dreapta-radacina-stanga;

- calculul inaltimii unui arbore binar;

- calculul numarului de noduri ale unui arbore binar;

- calculul informatiei maxime din nodurile unui arbore binar;

- operatie pentru a determina daca un arbore binar are

proprietatea ca orice nod al sau care nu e frunza are doi fii;

- operatie pentru a determina daca un arbore binar este echilibrat,

adica orice nod al sau are diferenta dintre inaltimea subarborelui

stang si inaltimea subarborelui drept egala cu 0, 1 sau -1. )

endfm

fmod TEMA16 is \*\*\*> 150 puncte

\*\*\*( Operatii cu arbori binari de cautare:

- o operatie care sa determine daca un arbore binar cu informatiile

din noduri numere naturale este arbore binar de cautare;

- o operatie care sa determine daca un numar natural se afla printre

valorile din nodurile unui arbore binar de cautare, fara a calcula

lista valorilor din nodurile arborelui respectiv;

- sortarea descrescatoare a unei liste de numere naturale, folosind

un arbore binar de cautare creat cu operatiile insert si insertlist

din modulul ARBBINCAUT de mai sus (exact cu definitiile de acolo);

- sa se sorteze crescator cu arbori binari de cautare liste de:

(a) numere intregi, folosind modulul predefinit INT;

(b) numere rationale, folosind modulul predefinit RAT;

(c) numere rationale, folosind modulul predefinit FLOAT;

(d) caractere, folosing modulul predefinit STRING;

(e) siruri de caractere, folosing modulul predefinit STRING. )

endfm

fmod TEMA17 is \*\*\*> 150 puncte

\*\*\*( Operatii cu arbori oarecare:

- calculul inaltimii unui arbore oarecare;

- calculul numarului de noduri ale unui arbore oarecare;

- calculul informatiei maxime din nodurile unui arbore oarecare;

- determinarea numarului maxim de fii dintre toate varfurile unui

arbore oarecare;

- parcurgerea in adancime a unui arbore oarecare, dar luand subarborii

fiecarui nod de la dreapta la stanga;

- parcurgerea unui arbore oarecare pe niveluri de la dreapta la stanga. )

endfm

**LABORATOR 10**

fmod TEMA12 is

protecting NAT .

sorts Multime MultMultime Pereche MultimePerechi .

subsorts Nat < Multime < MultMultime .

subsort Pereche < MultimePerechi .

op nil : -> Multime .

op \_\_ : Multime Multime -> Multime [assoc comm id: nil prec 20] .

op null : -> MultMultime .

op \_;\_ : MultMultime MultMultime -> MultMultime [assoc comm id: null prec 30] .

op (\_,\_) : Nat Nat -> Pereche .

op vida : -> MultimePerechi .

op \_|\_ : MultimePerechi MultimePerechi -> MultimePerechi [assoc comm id: vida prec 30] .

\*\*\*> concatenarea de multimi de perechi de numere naturale

op \_x\_ : Multime Multime -> MultimePerechi [prec 25] . \*\*\*> produsul cartezian

op adauga : Nat MultMultime -> MultMultime .

op parti : Multime -> MultMultime . \*\*\*> partile unei multimi

\*\*\*> trebuie sa primeasca o multime ca argument, adica o lista fara duplicate

vars X Y : Nat .

vars M N : Multime .

var P : MultMultime .

eq nil x M = vida .

eq M x nil = vida .

eq X M x Y N = (X,Y) | X x N | M x Y | M x N .

eq adauga(X,null) = null .

eq adauga(X,M ; P) = X M ; adauga(X,P) .

eq parti(nil) = nil . \*\*\*> Atentie: nu null!

eq parti(X M) = parti(M) ; adauga(X,parti(M)) .

endfm

fmod TEMA13 is

\*\*\*> adunarea, scaderea si inmultirea polinoamelor cu coeficienti intregi

protecting INT .

sorts Monom Polinom .

subsort Monom < Polinom .

op (\_,\_) : Int Nat -> Monom . \*\*\*> un monom = (coeficient,exponent)

\*\*\*> presupunem monoamele dintr-un polinom asezate descrescator dupa exponent

op nil : -> Polinom .

op \_\_ : Polinom Polinom -> Polinom [assoc id: nil prec 20] .

ops \_+\_ : Polinom Polinom -> Polinom [assoc comm prec 40] .

op \_\*\_ : Polinom Polinom -> Polinom [assoc comm prec 30] .

op \_-\_ : Polinom Polinom -> Polinom [prec 40] .

ops p1 p2 p3 : -> Polinom .

vars C1 C2 : Int .

vars E1 E2 : Nat .

vars P1 P2 : Polinom .

ceq (C1,E1) P1 + (C2,E2) P2 = (C1 + C2,E1) (P1 + P2) if E1 == E2 .

ceq (C1,E1) P1 + (C2,E2) P2 = (C1,E1) (P1 + (C2,E2) P2) if E1 > E2 .

ceq (C1,E1) P1 + (C2,E2) P2 = (C2,E2) ((C1,E1) P1 + P2) if E1 < E2 .

eq nil + P1 = P1 .

ceq (C1,E1) P1 - (C2,E2) P2 = (C1 - C2,E1) (P1 - P2) if E1 == E2 .

ceq (C1,E1) P1 - (C2,E2) P2 = (C1,E1) (P1 - (C2,E2) P2) if E1 > E2 .

ceq (C1,E1) P1 - (C2,E2) P2 = (- C2,E2) ((C1,E1) P1 - P2) if E1 < E2 .

eq P1 - nil = P1 .

eq nil - P1 = (0,0) - P1 .

eq (C1,E1) P1 \* (C2,E2) P2 = (C1 \* C2,E1 + E2) + P1 \* P2 + (C1,E1) \* P2 + (C2,E2) \* P1 .

eq P1 \* nil = nil .

eq p1 = (2,3) (7,1) (5,0) .

eq p2 = (1,2) (1,0) .

eq p3 = (2,1) .

endfm

fmod TEMA14 is

sorts Bit SirBit .

subsort Bit < SirBit .

ops 0 1 : -> Bit .

op nil : -> SirBit .

op \_\_ : SirBit SirBit -> SirBit [assoc id: nil prec 20] .

op \_andb\_ : Bit Bit -> Bit .

op \_orb\_ : Bit Bit -> Bit .

op \_xorb\_ : Bit Bit -> Bit .

op \_and\_ : SirBit SirBit -> SirBit [prec 30] .

op \_or\_ : SirBit SirBit -> SirBit [prec 30] .

op \_xor\_ : SirBit SirBit -> SirBit [prec 30] .

op \_+\_ : SirBit SirBit -> SirBit [prec 30] .

op t : SirBit -> SirBit . \*\*\*> transport

vars A B : Bit .

vars S T : SirBit .

eq 0 andb B = 0 .

eq B andb 0 = 0 .

eq 1 andb 1 = 1 .

eq 1 orb B = 1 .

eq B orb 1 = 1 .

eq 0 orb 0 = 0 .

eq A xorb B = if A == B then 0 else 1 fi .

eq nil and S = S .

eq S and nil = S .

eq S A and T B = (S and T) (A andb B) .

eq nil or S = S .

eq S or nil = S .

eq S A or T B = (S or T) (A orb B) .

eq nil xor S = S .

eq S xor nil = S .

eq S A xor T B = (S xor T) (A xorb B) .

eq S + nil = S .

eq nil + S = S .

eq S 0 + T 0 = (S + T) 0 .

eq S 0 + T 1 = (S + T) 1 .

eq S 1 + T 0 = (S + T) 1 .

eq S 1 + T 1 = t(S + T) 0 .

eq t(nil) = 1 .

eq t(S 0) = S 1 .

eq t(S 1) = t(S) 0 .

endfm

fmod TEMA15 is

\*\*\*> arbori binari

extending NAT .

sorts Lista Arbbin .

subsort Nat < Lista .

op nil : -> Lista . \*\*\*> lista de numere naturale vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbbin . \*\*\*> arborele binar vid

op \_{\_,\_} : Nat Arbbin Arbbin -> Arbbin .

op drs : Arbbin -> Lista .

\*\*\*> parcurgerea unui arbore binar in ordinea dreapta-radacina-stanga

ops h nr maxim : Arbbin -> Nat .

\*\*\*> inaltimea, numarul de noduri, maximul dintre informatiile din noduri

ops 2fii ech : Arbbin -> Bool .

\*\*\*> 2fii determina daca orice nod care nu e frunza are 2 fii

\*\*\*> ech determina daca arborele binar este echilibrat

op test : Nat Nat -> Bool .

\*\*\*> operatie auxiliara pentru ech

ops ex1 ex2 ex3 : -> Arbbin . \*\*\*> exemple de arbori binari

vars X Y : Nat .

vars A B : Arbbin .

eq drs(null) = nil .

eq drs(X{A,B}) = drs(B) X drs(A) .

eq h(null) = 0 .

eq h(X{A,B}) = s max(h(A),h(B)) .

eq nr(null) = 0 .

eq nr(X{A,B}) = s(nr(A) + nr(B)) .

eq maxim(null) = 0 .

eq maxim(X{A,B}) = max(X,max(maxim(A),maxim(B))) .

eq 2fii(null) = true .

eq 2fii(X{null,null}) = true .

ceq 2fii(X{A,null}) = false if A =/= null .

ceq 2fii(X{null,B}) = false if B =/= null .

ceq 2fii(X{A,B}) = 2fii(A) and 2fii(B) if A =/= null and B =/= null .

eq ech(null) = true .

eq ech(X{A,B}) = if test(h(A),h(B)) then ech(A) and ech(B) else false fi .

eq test(X,Y) = X == Y or X == s Y or s X == Y .

eq ex1 = 1{2{null,null},3{null,4{null,null}}} .

eq ex2 = 1{2{4{null,null},5{6{null,null},null}},3{null,7{8{null,null},9{10{null,null},null}}}} .

eq ex3 = 1{2{4{null,null},5{null,null}},3{null,null}} .

endfm

fmod TEMA16NAT is

extending TEMA15 .

op ecaut : Arbbin -> Bool .

\*\*\*> determina daca un arbore binar e arbore de cautare

ops minarbbin maxarbbin : Arbbin -> Nat .

\*\*\*( determina valoarea minima, respectiv maxima,

din nodurile unui arbore binar nevid )

op \_in\_ : Nat Arbbin -> Bool .

\*\*\*( determina daca o valoare se afla intr-un

arbore binar de cautare )

op insert : Nat Arbbin -> Arbbin .

op insertlist : Lista Arbbin -> Arbbin .

op sortdown : Lista -> Lista .

\*\*\*> sorteaza descrescator o lista

ops exarbcaut exnonarbcaut : -> Arbbin .

vars X Y Z : Nat .

var L : Lista .

vars A B C D : Arbbin .

eq minarbbin(X{null,null}) = X .

eq minarbbin(X{Y{A,B},null}) = min(X,minarbbin(Y{A,B})) .

eq minarbbin(X{null,Z{C,D}}) = min(X,minarbbin(Z{C,D})) .

eq minarbbin(X{Y{A,B},Z{C,D}}) = min(X,min(minarbbin(Y{A,B}),minarbbin(Z{C,D}))) .

eq maxarbbin(X{null,null}) = X .

eq maxarbbin(X{Y{A,B},null}) = max(X,maxarbbin(Y{A,B})) .

eq maxarbbin(X{null,Z{C,D}}) = max(X,maxarbbin(Z{C,D})) .

eq maxarbbin(X{Y{A,B},Z{C,D}}) = max(X,max(maxarbbin(Y{A,B}),maxarbbin(Z{C,D}))) .

eq ecaut(null) = true .

eq ecaut(X{null,null}) = true .

eq ecaut(X{Y{A,B},null}) = maxarbbin(Y{A,B}) <= X and ecaut(Y{A,B}) .

eq ecaut(X{null,Z{C,D}}) = X < minarbbin(Z{C,D}) and ecaut(Z{C,D}) .

eq ecaut(X{Y{A,B},Z{C,D}}) = maxarbbin(Y{A,B}) <= X and X < minarbbin(Z{C,D}) and ecaut(Y{A,B}) and ecaut(Z{C,D}) .

eq X in null = false .

eq X in X{A,B} = true .

ceq X in Y{A,B} = X in A if X < Y .

ceq X in Y{A,B} = X in B if X > Y .

eq insert(Y,null) = Y{null,null} .

eq insert(Y,X{A,B}) = if Y <= X then X{insert(Y,A),B} else X{A,insert(Y,B)} fi .

eq insertlist(nil,A) = A .

eq insertlist(X L,A) = insert(X,insertlist(L,A)) .

eq sortdown(L) = drs(insertlist(L,null)) .

eq exarbcaut = 10{2{null,null},20{15{null,null},100{null,null}}} .

eq exnonarbcaut = 10{2{null,null},20{1{null,null},100{null,null}}} .

\*\*\*> red ecaut(exarbcaut) . => true.

\*\*\*> red ecaut(exnonarbcaut) . => false.

endfm

fmod TEMA16a is

protecting INT .

sorts Lista Arbbin .

subsort Int < Lista .

op nil : -> Lista . \*\*\*> lista de numere intregi vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbbin . \*\*\*> arborele binar vid

op \_{\_,\_} : Int Arbbin Arbbin -> Arbbin .

op ino : Arbbin -> Lista .

\*\*\*> parcurgerea unui arbore binar in inordine

op insert : Int Arbbin -> Arbbin .

op insertlist : Lista Arbbin -> Arbbin .

op sortare : Lista -> Lista .

\*\*\*> sorteaza o lista de intregi

vars X Y Z : Int .

var L : Lista .

vars A B C D : Arbbin .

eq ino(null) = nil .

eq ino(X{A,B}) = ino(A) X ino(B) .

eq insert(Y,null) = Y{null,null} .

eq insert(Y,X{A,B}) = if Y <= X then X{insert(Y,A),B} else X{A,insert(Y,B)} fi .

eq insertlist(nil,A) = A .

eq insertlist(X L,A) = insert(X,insertlist(L,A)) .

eq sortare(L) = ino(insertlist(L,null)) .

endfm

fmod TEMA16b is

protecting RAT .

sorts Lista Arbbin .

subsort Rat < Lista .

op nil : -> Lista . \*\*\*> lista de numere rationale vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbbin . \*\*\*> arborele binar vid

op \_{\_,\_} : Rat Arbbin Arbbin -> Arbbin .

op ino : Arbbin -> Lista .

\*\*\*> parcurgerea unui arbore binar in inordine

op insert : Rat Arbbin -> Arbbin .

op insertlist : Lista Arbbin -> Arbbin .

op sortare : Lista -> Lista .

\*\*\*> sorteaza o lista de numere rationale

vars X Y Z : Rat .

var L : Lista .

vars A B C D : Arbbin .

eq ino(null) = nil .

eq ino(X{A,B}) = ino(A) X ino(B) .

eq insert(Y,null) = Y{null,null} .

eq insert(Y,X{A,B}) = if Y <= X then X{insert(Y,A),B} else X{A,insert(Y,B)} fi .

eq insertlist(nil,A) = A .

eq insertlist(X L,A) = insert(X,insertlist(L,A)) .

eq sortare(L) = ino(insertlist(L,null)) .

endfm

fmod TEMA16c is

protecting FLOAT .

sorts Lista Arbbin .

subsort Float < Lista .

op nil : -> Lista . \*\*\*> lista de numere reale vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbbin . \*\*\*> arborele binar vid

op \_{\_,\_} : Float Arbbin Arbbin -> Arbbin .

op ino : Arbbin -> Lista .

\*\*\*> parcurgerea unui arbore binar in inordine

op insert : Float Arbbin -> Arbbin .

op insertlist : Lista Arbbin -> Arbbin .

op sortare : Lista -> Lista .

\*\*\*> sorteaza o lista de numere reale

vars X Y Z : Float .

var L : Lista .

vars A B C D : Arbbin .

eq ino(null) = nil .

eq ino(X{A,B}) = ino(A) X ino(B) .

eq insert(Y,null) = Y{null,null} .

eq insert(Y,X{A,B}) = if Y <= X then X{insert(Y,A),B} else X{A,insert(Y,B)} fi .

eq insertlist(nil,A) = A .

eq insertlist(X L,A) = insert(X,insertlist(L,A)) .

eq sortare(L) = ino(insertlist(L,null)) .

endfm

fmod TEMA16d is

protecting STRING .

sorts Lista Arbbin .

subsort Char < Lista .

op nil : -> Lista . \*\*\*> lista de caractere vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbbin . \*\*\*> arborele binar vid

op \_{\_,\_} : Char Arbbin Arbbin -> Arbbin .

op ino : Arbbin -> Lista .

\*\*\*> parcurgerea unui arbore binar in inordine

op insert : Char Arbbin -> Arbbin .

op insertlist : Lista Arbbin -> Arbbin .

op sortare : Lista -> Lista .

\*\*\*> sorteaza o lista de caractere

vars X Y Z : Char .

var L : Lista .

vars A B C D : Arbbin .

eq ino(null) = nil .

eq ino(X{A,B}) = ino(A) X ino(B) .

eq insert(Y,null) = Y{null,null} .

eq insert(Y,X{A,B}) = if Y <= X then X{insert(Y,A),B} else X{A,insert(Y,B)} fi .

eq insertlist(nil,A) = A .

eq insertlist(X L,A) = insert(X,insertlist(L,A)) .

eq sortare(L) = ino(insertlist(L,null)) .

endfm

fmod TEMA16e is

protecting STRING .

sorts Lista Arbbin .

subsort String < Lista .

op nil : -> Lista . \*\*\*> lista de siruri de caractere vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbbin . \*\*\*> arborele binar vid

op \_{\_,\_} : String Arbbin Arbbin -> Arbbin .

op ino : Arbbin -> Lista .

\*\*\*> parcurgerea unui arbore binar in inordine

op insert : String Arbbin -> Arbbin .

op insertlist : Lista Arbbin -> Arbbin .

op sortare : Lista -> Lista .

\*\*\*> sorteaza o lista de siruri de caractere

vars X Y Z : String .

var L : Lista .

vars A B C D : Arbbin .

eq ino(null) = nil .

eq ino(X{A,B}) = ino(A) X ino(B) .

eq insert(Y,null) = Y{null,null} .

eq insert(Y,X{A,B}) = if Y <= X then X{insert(Y,A),B} else X{A,insert(Y,B)} fi .

eq insertlist(nil,A) = A .

eq insertlist(X L,A) = insert(X,insertlist(L,A)) .

eq sortare(L) = ino(insertlist(L,null)) .

endfm

fmod ARBORE is

\*\*\*> arbori oarecare

protecting NAT .

sorts Lista Arbore ListArb .

subsort Nat < Lista .

subsort Arbore < ListArb .

op nil : -> Lista . \*\*\*> lista de numere naturale vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbore . \*\*\*> arborele vid

op \_{\_} : Nat ListArb -> Arbore [prec 20] .

op frunza : -> ListArb . \*\*\*> lista de arbori vida

op \_;\_ : ListArb ListArb -> ListArb [assoc id: frunza prec 30] .

\*\*\*> concatenarea de liste de arbori

op ex : -> Arbore . \*\*\*> un exemplu de arbore oarecare

eq ex = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza}} ; 7{frunza}} .

endfm

fmod TEMA17 is

protecting ARBORE .

ops maxlist : Lista -> Nat .

\*\*\*> maximul dintr-o lista de numere naturale

op lungime : ListArb -> Nat .

\*\*\*> lungimea unei liste de arbori

ops h nr maxim nrfii nrmaxfii : Arbore -> Nat .

\*\*\*( inaltimea, numarul de noduri, valoarea maxima din noduri, numarul de

fii, respectiv numarul maxim de fii al unui nod intr-un arbore oarecare )

ops hlist nrlist maximlist nrfiilist : Arbore -> Lista .

\*\*\*> operatii auxiliare pentru h, nr, maxim, respectiv nrmaxfii

ops dfds bfds : Arbore -> Lista .

\*\*\*> parcurgerile in adancime, respectiv pe niveluri, de la dreapta la stanga

ops dfdslist bfdslist : ListArb -> Lista .

var X : Nat .

var L : Lista .

var A : Arbore .

vars LA LA1 : ListArb .

eq maxlist(nil) = 0 .

eq maxlist(X L) = max(X,maxlist(L)) .

eq lungime(frunza) = 0 .

eq lungime(A ; LA) = s lungime(LA) .

eq h(null) = 0 .

eq h(X{LA}) = s maxlist(hlist(LA)) .

eq hlist(frunza) = nil .

eq hlist(A ; LA) = h(A) hlist(LA) .

eq nr(null) = 0 .

eq nr(X{LA}) = s nrlist(LA) .

eq nrlist(frunza) = 0 .

eq nrlist(A ; LA) = nr(A) + nrlist(LA) .

eq maxim(null) = 0 .

eq maxim(X{LA}) = max(X,maximlist(LA)) .

eq maximlist(frunza) = 0 .

eq maximlist(A ; LA) = max(maxim(A),maximlist(LA)) .

eq nrfii(null) = 0 .

eq nrfii(X{LA}) = lungime(LA) .

eq nrfiilist(frunza) = nil .

eq nrfiilist(A ; LA) = nrfii(A) nrfiilist(LA) .

eq nrmaxfii(null) = 0 .

eq nrmaxfii(X{LA}) = max(nrfii(X{LA}),maxlist(nrfiilist(LA))) .

eq dfds(null) = nil .

eq dfds(X{LA}) = X dfdslist(LA) .

eq dfdslist(frunza) = nil .

eq dfdslist(LA ; A) = dfds(A) dfdslist(LA) .

eq bfds(null) = nil .

eq bfds(X{LA}) = X bfdslist(LA) .

eq bfdslist(frunza) = nil .

eq bfdslist(LA1 ; X{LA}) = X bfdslist(LA ; LA1) .

endfm

fmod TEMA-OBLIGATORIE is \*\*\*> pentru bonus la nota la prima lucrare

\*\*\*( Exercitiile din aceasta tema sunt obligatorii, in sensul

ca rezolvarea lor conteaza pentru notarea la prima lucrare de

control. Rezolvarile acestor exercitii trebuie aduse de

studentii fiecarei grupe la primul laborator de dupa Paste.

Daca sunt aduse mai tarziu, atunci nu vor fi notate. )

\*\*\*( Prima cerinta din tema obligatorie este sa scrieti un

modul in Maude pentru numere complexe, care sa importe modulul

predefinit FLOAT, in care sunt definite numerele reale, pentru

lucrul cu partea reala si partea imaginara a unui numar complex.

Modulul vostru pentru numere complexe trebuie sa contina

operatiile de: calcul al modulului unui numar complex, adunare,

scadere, inmultire si impartire a doua numere complexe. )

\*\*\*( A doua cerinta din tema obligatorie este sa scrieti in Maude

un modul pentru polinoame cu coeficienti intregi, care sa contina

o operatie pentru determinarea listei tuturor radacinilor intregi

ale unui polinom cu coeficienti intregi. Amintesc ca toate

radacinile intregi ale unui polinom cu coeficienti intregi sunt

divizori intregi ai termenului liber al polinomului. A se revedea

TEMA 13 din modlectia7\_8.maude, a carei rezolvare am sa v-o trimit

la sfarsitul acestei saptamani. )

\*\*\*( Ultima cerinta din tema obligatorie este un exercitiu dat de

doamna Profesoara I. Leustean, care va cere sa scrieti in Maude un

modul pentru implementarea tipului multiset cu elemente din

multimea numerelor naturale.

Daca E este o multime, atunci un multiset cu elemente din E

este o multime de perechi M = {(x,n) | x in E, n natural}. Intr-un

multiset, un element poate aparea de mai multe ori: (x,n) in M

inseamna ca x apare de n ori in M. Un multiset poate fi scris ca o

multime in care permitem ca un element sa apara de mai multe ori

(ordinea elementelor nu are importanta), sau ca o lista de perechi,

cu prima componenta dintr-o pereche reprezentand elementul, iar a

doua componenta a perechii reprezentand numarul de aparitii ale

elementului in multiset. De exemplu: M = {1, 2, 1, 5, 3, 4, 5, 4, 1,

3, 5, 0} = {(1,3), (2,1), (3,2), (4,2), (5,3), (0,1)}.

Scrieti un modul pentru multiseturi cu elemente din multimea

numerelor naturale, care sa contina doua operatii pentru trecerea

dintr-o reprezentare in cealalta, la care sa adaugati alte doua

operatii, pentru definirea reuniunii si, respectiv, a intersectiei

a doua multiseturi in reprezentarea cu perechi. In reuniunea a doua

multiseturi, fiecare element are ca numar de aparitii maximul dintre

numerele aparitiilor sale in fiecare dintre cele doua multiseturi, in

timp ce, in intersectia a doua multiseturi, fiecare element are ca

numar de aparitii minimul dintre numerele aparitiilor sale in fiecare

dintre cele doua multiseturi.

Ca indicatie, multiseturile in prima reprezentare vor fi liste de

numere naturale, avand concatenarea comutativa si admitand duplicate,

iar multiseturile in a doua reprezentare vor fi liste de perechi de

numere naturale, avand concatenarea comutativa si admitand cel mult

o pereche cu o anumita prima componenta. )

endfm

fmod OBSERVATIE-TEMA is

\*\*\*( Si temele urmatoare, care nu sunt obligatorii, trebuie aduse tot

la primul laborator de dupa Paste. Amintesc ca punctajele acumulate

din rezolvarea acestor teme, pe care vi le dau dupa fiecare laborator,

va pot furniza un punctaj-bonus la examen. )

endfm

fmod TEMA18 is \*\*\*> 150 puncte

\*\*\*( Scrieti un modul pentru polinoame cu coeficienti reali, importand

modulul FLOAT predefinit, care sa contina operatii pentru adunarea,

scaderea, inmultirea a doua polinoame, precum si pentru catul si restul

impartirii a doua polinoame cu coeficienti reali. )

endfm

fmod TEMA19 is \*\*\*> 90 puncte

\*\*\*( Scrieti un modul pentru implementarea unui automat finit

determinist, cu starile, starea initiala, starile finale, alfabetul

si functia de tranzitie date in interiorul modulului, si care sa

contina o operatie care sa determine daca un cuvant este sau nu

acceptat de automat. )

endfm

fmod TEMA20 is \*\*\*> 90 puncte

\*\*\*> La fel ca in TEMA 19, dar pentru un automat finit nedeterminist.

endfm

**LABORATOR 11\_12**

fmod OBSERVATIE-LUCRARE-2 is

\*\*\*( A se vedea, la sfarsitul acestui fisier, exercitiile

pregatitoare pentru a doua lucrare de control. )

endfm

fmod GRAF is

\*\*\*( parcurgerile grafurilor orientate sau neorientate

(dupa cum dam listele de vecini ale fiecarui varf) )

\*\*\*( probleme fata de parcurgerile arborilor: posibila neconexitate,

posibila neunicitate a drumurilor, datorata prezentei ciclurilor )

protecting NAT .

sorts Lista VarfVecini Graf .

subsort Nat < Lista .

subsort VarfVecini < Graf .

op nil : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc id: nil prec 20] .

op \_{\_} : Nat Lista -> VarfVecini [prec 25] .

op null : -> Graf . \*\*\*> graful vid

op \_;\_ : Graf Graf -> Graf [assoc id: null prec 30] .

\*\*\*> un graf este o lista de varfuri urmate de listele vecinilor lor

op \_in\_ : Nat Lista -> Bool [prec 40] .

\*\*\*( operatia apartine; va fi folosita in locul unui vector in care

sa fie marcate varfurile vizitate )

ops bf df : Graf -> Lista .

\*\*\*> parcurgerile in latime, respectiv in adancime

\*\*\*( parcurgerile vor incepe intotdeauna cu primul varf

din lista care da graful )

ops bfaux dfaux : Graf Graf Lista -> Lista .

op transf : Lista Graf -> Graf .

op cauta : Nat Graf -> Graf .

op peniveluri : Lista Graf Lista -> Lista .

op cautavecini : Nat Graf -> Lista .

\*\*\*> operatii auxiliare pentru implementarea operatiilor de parcurgere

ops exno exo : -> Graf .

\*\*\*> un exemplu de graf neorientat si unul de graf orientat

vars G H : Graf .

vars X Y : Nat .

vars V L : Lista .

eq X in nil = false .

eq X in Y L = if X == Y then true else X in L fi .

eq df(G) = dfaux(G,G,nil) .

eq dfaux(null,G,L) = L .

eq dfaux(X{V} ; H,G,L) = if X in L then dfaux(H,G,L) else dfaux(transf(V,G) ; H,G,L X) fi .

eq transf(nil,G) = null .

eq transf(X L,G) = cauta(X,G) ; transf(L,G) .

eq cauta(X,null) = null .

eq cauta(X,Y{V} ; G) = if X == Y then X{V} else cauta(X,G) fi .

eq bf(G) = bfaux(G,G,nil) .

eq bfaux(null,G,L) = L .

eq bfaux(X{V} ; H,G,L) = if X in L then bfaux(H,G,L) else bfaux(H,G,L X peniveluri(V,G,L X)) fi .

eq peniveluri(nil,G,L) = nil .

eq peniveluri(X V,G,L) = if X in L then peniveluri(V,G,L) else X peniveluri(V cautavecini(X,G),G,L X) fi .

eq cautavecini(X,null) = nil .

eq cautavecini(X,Y{V} ; G) = if X == Y then V else cautavecini(X,G) fi .

eq exno = 1{2 3} ; 2{1 3 6} ; 3{1 2 4 8 10} ; 4{3} ; 5{nil} ; 6{2 7} ; 7{6} ; 8{3 9} ; 9{8} ; 10{nil} .

eq exo = 1{2} ; 2{3} ; 3{1 4 8 10} ; 4{nil} ; 5{nil} ; 6{2} ; 7{6} ; 8{9} ; 9{nil} ; 10{nil} .

endfm

fmod CIUR is \*\*\*> ciurul lui Eratostene

protecting NAT .

sort Lista .

subsort Nat < Lista .

op nil : -> Lista .

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op filtreaza\_cu\_ : Lista Nat -> Lista .

op mai-mici-decat\_ : Nat -> Lista .

op ciuruire\_ : Lista -> Lista .

op primele-pana-la\_ : Nat -> Lista .

var X : Nat .

vars P N : NzNat .

var L : Lista .

eq filtreaza nil cu P = nil .

eq filtreaza (N L) cu P = if P divides N then filtreaza L cu P else N filtreaza L cu P fi .

eq mai-mici-decat 0 = nil .

eq mai-mici-decat 1 = nil .

eq mai-mici-decat 2 = 2 .

eq mai-mici-decat s s N = mai-mici-decat s N s s N .

eq ciuruire nil = nil .

eq ciuruire (N L) = N (ciuruire (filtreaza L cu N)) .

eq primele-pana-la X = ciuruire (mai-mici-decat X) .

endfm

fmod OBSERVATIE-TEME is

\*\*\*( Temele urmatoare au fost date inainte de Paste. Le puteti aduce

si saptamana viitoare, dar nu mai tarziu de saptamana viitoare. Daca

le aduceti in ultima saptamana a semestrului, atunci nu vi le punctez.

Dintre aceste teme, cea obligatorie este pentru un bonus la nota pe

care o obtineti la prima lucrare de control.

Celelalte sunt, dupa cum stiti, pentru un bonus la nota de la examen.

Va rog sa cititi lectiile, temele si subiectele tip lucrare cu arbori

binari si arbori oarecare pe care vi le-am trimis, ca pregatire pentru

a doua lucrare de control. )

endfm

fmod TEMA-OBLIGATORIE is \*\*\*> pentru bonus la nota la prima lucrare

\*\*\*( Exercitiile din aceasta tema sunt obligatorii, in sensul

ca rezolvarea lor conteaza pentru notarea la prima lucrare de

control. Rezolvarile acestor exercitii trebuie aduse de

studentii fiecarei grupe la primul laborator de dupa Paste.

Daca sunt aduse mai tarziu, atunci nu vor fi notate. )

\*\*\*( Prima cerinta din tema obligatorie este sa scrieti un

modul in Maude pentru numere complexe, care sa importe modulul

predefinit FLOAT, in care sunt definite numerele reale, pentru

lucrul cu partea reala si partea imaginara a unui numar complex.

Modulul vostru pentru numere complexe trebuie sa contina

operatiile de: calcul al modulului unui numar complex, adunare,

scadere, inmultire si impartire a doua numere complexe. )

\*\*\*( A doua cerinta din tema obligatorie este sa scrieti in Maude

un modul pentru polinoame cu coeficienti intregi, care sa contina

o operatie pentru determinarea listei tuturor radacinilor intregi

ale unui polinom cu coeficienti intregi. Amintesc ca toate

radacinile intregi ale unui polinom cu coeficienti intregi sunt

divizori intregi ai termenului liber al polinomului. A se revedea

TEMA 13 din modlectia7\_8.maude, a carei rezolvare am sa v-o trimit

la sfarsitul acestei saptamani. )

\*\*\*( Ultima cerinta din tema obligatorie este un exercitiu dat de

doamna Profesoara I. Leustean, care va cere sa scrieti in Maude un

modul pentru implementarea tipului multiset cu elemente din

multimea numerelor naturale.

Daca E este o multime, atunci un multiset cu elemente din E

este o multime de perechi M = {(x,n) | x in E, n natural}. Intr-un

multiset, un element poate aparea de mai multe ori: (x,n) in M

inseamna ca x apare de n ori in M. Un multiset poate fi scris ca o

multime in care permitem ca un element sa apara de mai multe ori

(ordinea elementelor nu are importanta), sau ca o lista de perechi,

cu prima componenta dintr-o pereche reprezentand elementul, iar a

doua componenta a perechii reprezentand numarul de aparitii ale

elementului in multiset. De exemplu: M = {1, 2, 1, 5, 3, 4, 5, 4, 1,

3, 5, 0} = {(1,3), (2,1), (3,2), (4,2), (5,3), (0,1)}.

Scrieti un modul pentru multiseturi cu elemente din multimea

numerelor naturale, care sa contina doua operatii pentru trecerea

dintr-o reprezentare in cealalta, la care sa adaugati alte doua

operatii, pentru definirea reuniunii si, respectiv, a intersectiei

a doua multiseturi in reprezentarea cu perechi. In reuniunea a doua

multiseturi, fiecare element are ca numar de aparitii maximul dintre

numerele aparitiilor sale in fiecare dintre cele doua multiseturi, in

timp ce, in intersectia a doua multiseturi, fiecare element are ca

numar de aparitii minimul dintre numerele aparitiilor sale in fiecare

dintre cele doua multiseturi.

Ca indicatie, multiseturile in prima reprezentare vor fi liste de

numere naturale, avand concatenarea comutativa si admitand duplicate,

iar multiseturile in a doua reprezentare vor fi liste de perechi de

numere naturale, avand concatenarea comutativa si admitand cel mult

o pereche cu o anumita prima componenta. )

endfm

fmod TEMA18 is \*\*\*> 150 puncte

\*\*\*( Scrieti un modul pentru polinoame cu coeficienti reali, importand

modulul FLOAT predefinit, care sa contina operatii pentru adunarea,

scaderea, inmultirea a doua polinoame, precum si pentru catul si restul

impartirii a doua polinoame cu coeficienti reali. )

endfm

fmod TEMA19 is \*\*\*> 90 puncte

\*\*\*( Scrieti un modul pentru implementarea unui automat finit

determinist, cu starile, starea initiala, starile finale, alfabetul

si functia de tranzitie date in interiorul modulului, si care sa

contina o operatie care sa determine daca un cuvant este sau nu

acceptat de automat. )

endfm

fmod TEMA20 is \*\*\*> 90 puncte

\*\*\*> La fel ca in TEMA 19, dar pentru un automat finit nedeterminist.

endfm

fmod ENUNTURI is

\*\*\*( Cerintele din exercitiile de mai jos au fost date la lucrari de control in anii anteriori. Unele dintre ele au fost rezolvate in modlectia9.maude si modlectia10.maude.

In a doua lucrare de control pe care o veti da, veti avea un exercitiu cu arbori binari si unul cu arbori oarecare (desigur, cele doua exercitii vor avea mai putine cerinte decat exercitiile (I) si (II) de mai jos).

In exercitiile de mai jos, sorturile pentru arbori binari, arbori oarecare si liste de arbori oarecare, precum si operatiile care construiesc aceste sorturi, vor fi denumite si declarate la fel ca in modlectia9.maude.

(I) Sa se scrie in Maude un modul pentru arbori binari cu numere intregi ca informatii in noduri, care sa includa modulul INT predefinit si sa contina urmatoarele operatii, definite ca mai jos.

op maxdif : Arbbin -> Nat .

op ecomplet : Arbbin -> Bool .

op invers : Arbbin -> Arbbin .

Pentru orice arbore binar A:

maxdif(A) = maximul modulului diferentei dintre numarul de frunze al subarborelui stang si numarul de frunze al subarborelui drept ale unui nod din arborele binar A;

ecomplet(A) = true daca A este arbore binar complet, si false in caz contrar;

invers(A) = arborele binar care se obtine din A prin inversarea subarborelui stang cu subarborele drept al fiecarui nod al lui A.

(II) Sa se scrie in Maude un modul pentru arbori oarecare cu numere naturale ca informatii in noduri, care sa includa modulul NAT predefinit si sa contina urmatoarele operatii, definite ca mai jos.

ops nrmaxfii nrminfiinenul nrnodint nrfrunze nrvalpare nrcunrimparfii : Arbore -> Nat .

ops estebin descrescniv : Arbore -> Bool .

ops addist radpara infoh : Arbore -> Arbore .

Pentru orice arbore A:

nrmaxfii(A) = numarul maxim de fii ai unui nod din arborele A;

nrminfiinenul(A) = numarul minim de fii ai unui nod intern (adica nod care nu e frunza) din arborele A, cu conventia: nrminfiinenul(null) = 0 si nrminfiinenul(F) = 0 daca F este o frunza;

nrnodint(A) = numarul de noduri interne (adica noduri care nu sunt frunze) ale arborelui A;

nrfrunze(A) = numarul de frunze ale arborelui A;

nrvalpare(A) = numarul de noduri din arborele A care au informatia para;

nrcunrimparfii(A) = numarul de noduri din arborele A care au un numar impar de fii;

estebin(A) = true daca A este arbore binar (cu aceeasi radacina ca aceea din reprezentarea sa ca arbore oarecare), si false in caz contrar;

descrescniv(A) = true daca informatia din fiecare nod intern (adica nod care nu e frunza) al lui A este mai mare sau egala cu informatia din fiecare fiu al sau, si false in caz contrar;

addist(A) = arborele obtinut din A prin inlocuirea fiecarei informatii I din fiecare nod al sau cu I + D, unde D este distanta de la radacina arborelui A la acel nod, adica numarul de niveluri care despart radacina lui A de acel nod (D = 0 in cazul radacinii, D = 1 pentru fiecare fiu al radacinii etc.);

radpara(A) = arborele obtinut din A prin eliminarea tuturor subarborilor sai cu informatia din radacina data de un numar natural impar;

infoh(A) = arborele obtinut din A prin inlocuirea informatiei din fiecare nod cu inaltimea subarborelui lui A de radacina acel nod.

A se vedea rezolvarile cerintelor de mai sus in modulele urmatoare. )

endfm

fmod EXERCITIUL-I is

protecting INT .

sorts Lista Arbbin .

subsort Int < Lista .

\*\*\*> Lista e sortul pentru liste de numere intregi

\*\*\*> Arbbin e sortul pentru arbori binari

op nil : -> Lista . \*\*\*> lista de numere intregi vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbbin . \*\*\*> arborele binar vid

op \_{\_,\_} : Int Arbbin Arbbin -> Arbbin .

\*\*\*> operatiile care construiesc sortul Arbbin

ops maxdif nrfrunze absdif h : Arbbin -> Nat .

ops ecomplet : Arbbin -> Bool .

op invers : Arbbin -> Arbbin .

op plinlaniv : Arbbin Nat Nat -> Bool .

ops ex1 ex2 ex3 ex4 ex5 ex6 ex7 ex8 ex9 ex10 : -> Arbbin .

vars H L M N : Nat .

vars I J K : Int .

vars A B C D : Arbbin .

eq nrfrunze(null) = 0 .

eq nrfrunze(I{null,null}) = 1 .

eq nrfrunze(I{null,J{A,B}}) = nrfrunze(J{A,B}) .

eq nrfrunze(I{J{A,B},null}) = nrfrunze(J{A,B}) .

eq nrfrunze(I{J{A,B},K{C,D}}) = nrfrunze(J{A,B}) + nrfrunze(K{C,D}) .

eq absdif(null) = 0 .

eq absdif(I{A,B}) = abs(nrfrunze(B) - nrfrunze(A)) .

eq maxdif(null) = 0 .

eq maxdif(I{A,B}) = max(absdif(I{A,B}),max(absdif(A),absdif(B))) .

\*\*\*> max si abs sunt predefinite in modulul INT.

eq invers(null) = null .

eq invers(I{A,B}) = I{invers(B),invers(A)} .

eq h(null) = 0 .

eq h(I{A,B}) = s max(h(A),h(B)) .

eq ecomplet(A) = plinlaniv(A,h(A),1) .

ceq plinlaniv(I{A,null},H,L) = false if L < H .

ceq plinlaniv(I{null,A},H,L) = false if L < H .

ceq plinlaniv(I{J{A,B},K{C,D}},H,L) = plinlaniv(J{A,B},H,s L) and plinlaniv(K{C,D},H,s L) if L < H .

ceq plinlaniv(I{A,B},H,L) = true if L >= H .

eq plinlaniv(null,H,L) = true .

eq ex1 = 1{2{null,null},-3{null,4{null,null}}} .

eq ex2 = 1{2{null,null},3{-6{null,null},4{7{null,null},5{null,null}}}} .

eq ex3 = 1{null,3{6{null,null},4{7{null,null},5{null,null}}}} .

eq ex4 = 10{7{null,-8{null,null}},15{-12{null,null},-20{18{null,null},null}}} .

eq ex5 = 10{null,25{null,25{null,null}}} .

eq ex6 = 10{10{null,null},25{null,25{null,null}}} .

eq ex7 = 10{7{null,30{null,null}},15{12{null,null},20{18{null,null},null}}} .

eq ex8 = 1{2{null,null},3{null,4{null,null}}} .

eq ex9 = 1{-2{null,null},-3{6{null,null},-4{7{null,null},5{null,null}}}} .

eq ex10 = 1{2{4{null,null},5{null,null}},3{6{null,null},7{null,null}}} .

endfm

fmod EXERCITIUL-II is

protecting NAT .

sorts Lista Arbore ListArb .

subsort Nat < Lista .

subsort Arbore < ListArb .

\*\*\*> Lista e sortul pentru liste de numere naturale

\*\*\*> Arbore e sortul pentru arbori oarecare

\*\*\*> ListArb e sortul pentru liste de arbori oarecare

op nil : -> Lista . \*\*\*> lista de numere naturale vida

op \_\_ : Lista Lista -> Lista [assoc id: nil] .

op null : -> Arbore . \*\*\*> arborele vid

op \_{\_} : Nat ListArb -> Arbore [prec 20] .

\*\*\*> operatiile care construiesc sortul Arbore

op frunza : -> ListArb . \*\*\*> lista de arbori vida

op \_;\_ : ListArb ListArb -> ListArb [assoc id: frunza prec 30] .

\*\*\*> concatenarea de liste de arbori

ops nrmaxfii nrminfiinenul nrnodint nrfrunze nrvalpare nrcunrimparfii : Arbore -> Nat .

ops estebin descrescniv : Arbore -> Bool .

ops addist radpara infoh : Arbore -> Arbore .

op lungime : ListArb -> Nat .

ops maxlist minlist : Lista -> Nat .

op listfara0 : Lista -> Lista .

op nrfii : Arbore -> Nat .

op nrfiilist : ListArb -> Lista .

ops nrnodintlist nrfrunzelist nrvalparelist nrcunrimparfiilist : ListArb -> Nat .

op descrescnivcuval : Arbore Nat -> Bool .

op descrescnivcuvallist : ListArb Nat -> Bool .

op addistcuniv : Arbore Nat -> Arbore .

op addistcunivlist : ListArb Nat -> ListArb .

ops listaradpara listainfoh : ListArb -> ListArb .

op h : Arbore -> Nat .

op listah : ListArb -> Lista .

ops ex1 ex2 ex3 ex4 ex5 ex6 ex7 ex8 ex9 : -> Arbore .

vars M N P : Nat .

var L : Lista .

vars A B C : Arbore .

var LA LA1 : ListArb .

eq maxlist(nil) = 0 .

eq maxlist(N L) = max(N,maxlist(L)) .

eq minlist(N) = N .

eq minlist(N M L) = min(N,minlist(M L)) .

eq lungime(frunza) = 0 .

eq lungime(A ; LA) = s lungime(LA) .

eq listfara0(nil) = nil .

eq listfara0(0 L) = listfara0(L) .

ceq listfara0(N L) = N listfara0(L) if N =/= 0 .

eq nrfii(null) = 0 .

eq nrfii(N{LA}) = lungime(LA) .

eq nrfiilist(frunza) = nil .

eq nrfiilist(A ; LA) = nrfii(A) nrfiilist(LA) .

eq nrmaxfii(null) = 0 .

eq nrmaxfii(N{LA}) = max(nrfii(N{LA}),maxlist(nrfiilist(LA))) .

eq nrminfiinenul(null) = 0 .

eq nrminfiinenul(N{frunza}) = 0 .

eq nrminfiinenul(N{A ; LA}) = minlist(nrfii(N{A ; LA}) listfara0(nrfiilist(A ; LA))) .

eq nrnodint(null) = 0 .

eq nrnodint(N{frunza}) = 0 .

eq nrnodint(N{A ; LA}) = s(nrnodintlist(A ; LA)) .

eq nrnodintlist(frunza) = 0 .

eq nrnodintlist(A ; LA) = nrnodint(A) + nrnodintlist(LA) .

eq nrfrunze(null) = 0 .

eq nrfrunze(N{frunza}) = 1 .

eq nrfrunze(N{A ; LA}) = nrfrunzelist(A ; LA) .

eq nrfrunzelist(frunza) = 0 .

eq nrfrunzelist(A ; LA) = nrfrunze(A) + nrfrunzelist(LA) .

eq estebin(null) = true .

eq estebin(N{frunza}) = true .

eq estebin(N{A}) = estebin(A) .

eq estebin(N{A ; B}) = estebin(A) and estebin(B) .

eq estebin(N{A ; B ; C ; LA}) = false .

eq nrvalpare(null) = 0 .

ceq nrvalpare(N{LA}) = nrvalparelist(LA) if not (2 divides N) .

ceq nrvalpare(N{LA}) = s nrvalparelist(LA) if 2 divides N .

eq nrvalparelist(frunza) = 0 .

eq nrvalparelist(A ; LA) = nrvalpare(A) + nrvalparelist(LA) .

eq nrcunrimparfii(null) = 0 .

ceq nrcunrimparfii(N{LA}) = nrcunrimparfiilist(LA) if 2 divides lungime(LA) .

ceq nrcunrimparfii(N{LA}) = s nrcunrimparfiilist(LA) if not (2 divides lungime(LA)) .

eq nrcunrimparfiilist(frunza) = 0 .

eq nrcunrimparfiilist(A ; LA) = nrcunrimparfii(A) + nrcunrimparfiilist(LA) .

eq descrescniv(null) = true .

eq descrescniv(N{frunza}) = true .

eq descrescniv(N{A ; LA}) = descrescnivcuval(N{A ; LA},N) .

eq descrescnivcuval(null,N) = true .

eq descrescnivcuval(M{frunza},N) = N >= M .

eq descrescnivcuval(M{P{LA1} ; LA},N) = N >= M and descrescnivcuval(P{LA1},M) and descrescnivcuvallist(LA,M) .

eq descrescnivcuvallist(frunza,N) = true .

eq descrescnivcuvallist((A ; LA),N) = descrescnivcuval(A,N) and descrescnivcuvallist(LA,N) .

eq addist(A) = addistcuniv(A,0) .

eq addistcuniv(null,N) = null .

eq addistcuniv(M{LA},N) = (M + N){addistcunivlist(LA,s N)} .

eq addistcunivlist(frunza,N) = frunza .

eq addistcunivlist((A ; LA),N) = addistcuniv(A,N) ; addistcunivlist(LA,N) .

eq radpara(null) = null .

ceq radpara(N{LA}) = null if not (2 divides N) .

ceq radpara(N{LA}) = N{listaradpara(LA)} if 2 divides N .

eq listaradpara(frunza) = frunza .

ceq listaradpara(N{LA1} ; LA) = radpara(N{LA1}) ; listaradpara(LA) if 2 divides N .

ceq listaradpara(N{LA1} ; LA) = listaradpara(LA) if not (2 divides N) .

eq infoh(null) = null .

eq infoh(N{LA}) = h(N{LA}){listainfoh(LA)} .

eq listainfoh(frunza) = frunza .

eq listainfoh(A ; LA) = infoh(A) ; listainfoh(LA) .

eq h(null) = 0 .

eq h(N{LA}) = s maxlist(listah(LA)) .

eq listah(frunza) = nil .

eq listah(A ; LA) = h(A) listah(LA) .

eq ex1 = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza}} ; 7{frunza}} .

eq ex2 = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza}}} .

eq ex3 = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza} ; 7{8{frunza} ; 9{frunza} ; 10{frunza}}}} .

eq ex4 = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza}} ; 7{frunza}} .

eq ex5 = 10{2{1{frunza} ; 0{frunza}} ; 5{3{frunza}} ; 7{frunza}} .

eq ex6 = 10{2{1{frunza} ; 2{frunza}} ; 5{3{frunza}} ; 7{frunza}} .

eq ex7 = 3{4{frunza} ; 0{6{frunza} ; 8{2{frunza} ; 10{frunza} ; 20{frunza}}}} .

eq ex8 = 12{5{frunza} ; 0{13{frunza} ; 8{2{frunza} ; 7{frunza} ; 20{frunza}}} ; 22{25{frunza}}} .

eq ex9 = 12{5{frunza} ; 0{8{2{frunza} ; 7{frunza} ; 20{frunza}}} ; 22{25{frunza}}} .

endfm