Algoritmos y Estructuras de Datos II

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo práctico 1: Especificación

Lollapatuza

alias: JUSCIKTYMTQNAEURXVZQ

Integrante	LU	Correo electrónico
Bustos, Juan	19/22	juani8.bustos@gmail.com
Dominguez, Leonardo	285/22	leodomingue2016@gmail.com
Nandín, Matías	227/22	imatinandin@gmail.com
Marín, Candela Emilia	1405/21	canmarin17@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

TAD String es Secuencia(Letra)
TAD Producto es String
TAD Persona es String

1. TAD Menu

TAD Menu

igualdad observacional
$(\forall m, m' : \text{menu})\ (m =_{\text{obs}} m' \iff (\text{Dicc(m)} =_{\text{obs}} \text{Dicc(m')}))$

géneros menu

exporta Generadores, EstaEnMenu?, DevolverPrecio

usa Bool, Nat, Diccionario, Producto

observadores básicos
Dicc : menu $\longrightarrow$ dicc(Producto, nat)

generadores
NuevoMenu : dicc(Producto $\times$ nat) $\longrightarrow$ menu

otras operaciones
EstaEnMenu? : menu $\times$ Producto $\longrightarrow$ bool
DevolverPrecio : menu m $\times$ Producto p $\longrightarrow$ nat {EstaEnMenu?(m,p)}

axiomas $\forall m$: menu, $\forall prod$: Producto
Dicc(NuevoMenu(m)) $\equiv$ m
EstaEnMenu?(menu,prod) $\equiv$ def?(prod,Dicc(menu))
DevolverPrecio(menu,prod) $\equiv$ obtener(prod,Dicc(menu))

Fin TAD

2. TAD Stock

TAD Stock

igualdad observacional
$(\forall s, s' : \text{stock})\ (s =_{\text{obs}} s' \iff (\text{Dicc(s)} =_{\text{obs}} \text{Dicc(s')}))$

géneros stock

exporta Generadores, EstaEnListadoStock?, DevolverStock, HayStockSuficiente?

usa Bool, Nat, Diccionario, Producto

observadores básicos
Dicc : stock $\longrightarrow$ dicc(Producto, nat)

generadores
NuevoStock : dicc(Producto $\times$ nat) $\longrightarrow$ stock
AgregarStock : stock s $\times$ Producto p $\times$ nat n $\longrightarrow$ stock {EstaEnListadoStock?(s,p)}
SacarStock : stock s $\times$ Producto p $\times$ nat n $\longrightarrow$ stock
{EstaEnListadoStock?(s,p) $\wedge_{\text{L}}$ HayStockSuficiente?(s,p,n)}

otras operaciones
EstaEnListadoStock? : stock $\times$ Producto $\longrightarrow$ bool
DevolverStock : stock s $\times$ Producto p $\longrightarrow$ nat {EstaEnListadoStock?(s,p)}
HayStockSufiente? : stock s $\times$ Producto p $\times$ nat n $\longrightarrow$ bool {EstaEnListadoStock?(s,p)}

axiomas $\forall s$: stock, $\forall prod$: Producto, $\forall cant$: nat
Dicc(NuevoStock(s)) $\equiv$ s
Dicc(AgregarStock(s, prod, cant) $\equiv$ definir(prod, obtener(prod, Dicc(s)) + cant, borrar(prod, Dicc(s))
Dicc(SacarStock(s, prod, cant) $\equiv$ definir(prod, obtener(prod, Dicc(s)) - cant, borrar(prod, Dicc(s))
EstaEnListadoStock?(stock,prod) $\equiv$ def?(prod, Dicc(stock)
DevolverStock(stock,prod) $\equiv$ obtener(prod, Dicc(stock))

HayStockSuficiente?(stock,prod,cant) $\equiv$ DevolverStock(stock,prod) $\geq$ cant

Fin TAD

3. TAD ListaDeDescuentos

TAD ListaDeDescuentos

igualdad observacional

$(\forall d, d' : \text{listaDeDescuentos})\ (d =_{\text{obs}} d' \iff (\text{Dicc(d)} =_{\text{obs}} \text{Dicc(d')}))$

géneros listaDeDescuentos

exporta Generadores, TieneDesc?, DevolverDesc

usa Bool, Nat, Diccionario, Producto, Tupla

observadores básicos

Dicc : listaDeDescuentos $\longrightarrow$ dicc(tupla(Producto, nat), nat)

generadores

NuevosDescuentos : dicc(tupla(Producto $\times$ nat) $\times$ nat) *dic* $\longrightarrow$ listaDeDescuentos
{$(\forall p$:Producto)$((\forall n$:nat)(def?(<prod,n>, dic) $\Rightarrow_{\text{L}}$ (0 < obtener(<prod,n>, dic) < 100))}

otras operaciones

ClavesDesc : listaDeDescuentos $\longrightarrow$ secu(<Producto, nat>)
EncontrarCantDesc : secu(<Producto $\times$ nat>) $\times$ Producto $\times$ nat $\longrightarrow$ nat
TieneDesc? : listaDeDescuentos $\times$ Producto $\times$ nat $\longrightarrow$ bool
DevolverDesc : listaDeDescuentos *ls* $\times$ Producto *p* $\times$ nat *n* $\longrightarrow$ nat {TieneDesc?(ls, p, n)}

axiomas $\forall ls$: listaDeDescuentos, $\forall prod$: Producto, $\forall cant$: nat, $\forall secProd$: secu(tupla(Producto, nat))

Dicc(NuevosDescuentos(ls)) $\equiv$ ls
ClavesDesc(ls) $\equiv$ claves(Dicc(ls))
EncontrarCantDesc(secProd, prod, cant) $\equiv$ **if** *vacia?*(*secProd*) **then**
0
else
if Prim(secProd)$_0$ = prod)
then
if Prim(secProd)$_1$ $\leq$ cant
then
max (prim(secProd)$_1$,
EncontrarCantDesc(fin(secProd),prod,cant))
else
EncontrarCantDesc(fin(secProd),prod,cant)
fi
else
EncontrarCantDesc(fin(secProd), prod, cant)
fi
fi
TieneDesc?(listaDesc, prod, cant) $\equiv$ $\neg$(EncontrarCantDesc(ClavesDesc(listaDesc),prod,cant)=0?)
DevolverDesc(listaDesc, prod, cant) $\equiv$ obtener($\langle$prod,EncontrarCantDesc(ClavesDesc(listaDesc),
prod,cant)$\rangle$, Dicc(listaDesc))

Fin TAD

4. TAD PuestoDeComida

TAD PuestoDeComida

igualdad observacional

$$(\forall pt, pt' : \text{puesto}) \left(pt =_{\text{obs}} pt' \iff \begin{pmatrix} \text{Stock(pt)}=_{\text{obs}} \text{Stock(pt')} \wedge \\ \text{Menu(pt)}=_{\text{obs}} \text{Menu(pt')} \wedge \\ \text{Descuentos(pt)}=_{\text{obs}} \text{Descuentos(pt')} \end{pmatrix} \right)$$

géneros puesto

exporta Generadores, Observadores Básicos

usa Bool, Nat, Menu, ListaDeDescuento, Stock, Secuencia, Tupla

observadores básicos

Stock : puesto $\longrightarrow$ stock
Menu : puesto $\longrightarrow$ menu
Descuentos : puesto $\longrightarrow$ listaDeDescuentos

generadores

NuevoPuesto : stock s × menu m × listaDeDescuentos d $\longrightarrow$ puesto
{(∀pr: producto)(EstaEnMenu?(m,pr) $\Rightarrow_{\text{L}}$ EstaEnListadoStock?(m,pr)}
IncrementarStock : puesto pt × prod $producto$ × cant nat $\longrightarrow$ puesto {EstaEnMenu?(Menu(pt),prod)}
ReducirStock : puesto pt × prod $producto$ × cant nat $\longrightarrow$ puesto
{EstaEnMenu?(Menu(pt),prod) $\wedge_{\text{L}}$ HayStockSufiente?(Stock(pt),prod,cant)}

axiomas ∀$cant$: nat,∀s: stock, ∀m: menu, ∀d: listaDeDescuentos, ∀$prod$: Producto, ∀pt: puesto, ∀s: secu(<Producto, nat>)

,

Stock(NuevoPuesto(s,m,d) $\equiv$ s
Stock(ReducirStock(pt,prod,cant)) $\equiv$ SacarStock(Stock(pt),prod,cant)
Stock(IncrementarStock(pt,prod,cant)) $\equiv$ AgregarStock(Stock(pt),prod,cant)
Menu(NuevoPuesto(s,m,d)) $\equiv$ m
Menu(ReducirStock(pt,prod,cant)) $\equiv$ Menu(pt)
Menu(IncrementarStock(pt,prod,cant)) $\equiv$ Menu(pt)
Descuentos(NuevoPuesto(s,m,d)) $\equiv$ d
Descuentos(ReducirStock(pt,prod,cant)) $\equiv$ Descuentos(pt)
Descuentos(IncrementarStock(pt,prod,cant)) $\equiv$ Descuentos(pt)

Fin TAD

TAD Venta es tupla(puestoC:puesto, p:Persona, prod:Producto, cant:nat)

5. TAD Lollapatuza

TAD Lollapatuza

géneros lollapatuza

igualdad observacional

$$(\forall lolla, lolla' : \text{lollapatuza}) \left(lolla =_{\text{obs}} lolla' \iff \begin{pmatrix} \text{puestos(lolla)} =_{\text{obs}} \text{puestos(lolla')} \wedge \\ \text{Pesonas(lolla)} =_{\text{obs}} \text{Pesonas(lolla')} \wedge \\ \text{HistorialVenta(lolla)} =_{\text{obs}} \\ \text{HistorialVenta(lolla´)} \end{pmatrix} \right)$$

exporta Persona1, generadores, Personas

usa Conjunto, PuestoDeComida, Secuencias, Personas, Multiconjunto

observadores básicos

Puestos : lollapatuza $\longrightarrow$ conj(puesto)
Personas : lollapatuza $\longrightarrow$ conj(Persona)
HistorialVenta : lollapatuza $\longrightarrow$ multiconj(Venta)

generadores

CrearLolla : conj(puesto) $conjPC$ × conj(Persona) $\longrightarrow$ lollapatuza
$$\left\{ \begin{array}{l} (\forall PC1, PC2\text{:puesto})(((\text{PC1,PC2}) \in \text{conjPC}) \Rightarrow_{\text{L}} (\forall p\text{:Producto}) \\ ((\text{SeEncuentraEnMenu?(Menu(PC1),p)} \wedge \text{SeEncuentraEnMenu?(Menu(PC2),p)}) \Rightarrow_{\text{L}} \\ (\text{DevolverPrecio(Menu(PC1),p)} = \text{DevolverPrecio(Menu(PC2),p)})) \end{array} \right\}$$

RegistrarVenta : lollapatuza *lolla* × puesto *pt* × Persona *p* × Producto *prod* × nat *cant* $\longrightarrow$ lollapatuza
$\left\{\begin{array}{l}\text{p} \in \text{Personas(lolla)} \wedge \text{pt} \in \text{Puestos(lolla)} \wedge_{\text{L}} \text{EstaEnMenu?(Menu(pt),prod)} \wedge_{\text{L}} \\ \text{HayStockSuficiente?(Stock(pt),prod,cant)}\end{array}\right\}$

Hackear : lollapatuza *lolla* × Persona *p* × Producto *prod* $\longrightarrow$ lollapatuza
{($\exists$ V : Venta)(V $\in$ HistorialVenta(Lolla) $\wedge_{\text{L}}$ VentaHackeable?(V,p,prod))}

otras operaciones

DameVentaHack : multiconj(Venta) × Persona × Producto $\longrightarrow$ Venta

VentaHackeable? : Venta × Persona × Producto $\longrightarrow$ bool

CalcularGasto : Venta *v* $\longrightarrow$ nat {EstaEnMenu?(Menu(v.puestoC),v.prod)}

GastoPersona : multiconj(Venta) *c* × Persona *p* $\longrightarrow$ nat
{($\forall v$:Ventas)(v $\in$ c $\Rightarrow_{\text{L}}$ EstaEnMenu?(Menu(v.puestoC),v.prod)}

BuscarPersona1 : multiconj(Venta) *c* × conj(Personas) *cp* $\longrightarrow$ Persona
{$\neg$ $\emptyset$?(cp) $\wedge$ ($\forall v$:Ventas)(v $\in$ c $\Rightarrow_{\text{L}}$ EstaEnMenu?(Menu(v.puestoC),v.prod)}

persona1 : lollapatuza *lolla* $\longrightarrow$ Persona {$\neg$ $\emptyset$?(Personas(lolla)}

axiomas $\forall conjPC$: conj(puesto), $\forall lolla$: lollapatuza, $\forall cant$: nat, $\forall prod$: Producto, $\forall p$: Persona, $\forall conjP$: conj(Personas), $\forall pt$: puesto, $\forall v$: Venta

Puestos(CrearLolla(conjPC, conjP)) $\equiv$ conjPC

Puestos(RegistrarVenta(lolla, pt, p, prod, cant)) $\equiv$ (Puestos(lolla) - {pt}) $\cup$ {ReducirStock(pt,prod,cant)}

Puestos(Hackear(lolla, p, prod)) $\equiv$ (Puestos(lolla) - {DameVentaHack(HistorialVenta(lolla),p,prod).puestoC)}
$\cup$ {IncrementarStock(DameVentaHack(HistorialVenta(lolla),p,prod).puestoC,
prod, DameVentaHack(HistorialVenta(lolla),p,prod).cant}

Personas(CrearLolla(conjPC, ConjP)) $\equiv$ conjP

Personas(RegistrarVenta(lolla, pt, p, prod, cant)) $\equiv$ Personas(lolla)

Personas(Hackear(lolla, p, prod)) $\equiv$ Personas(lolla)

HistorialVenta(CrearLolla(conjPC, conjP)) $\equiv$ $\emptyset$

HistorialVenta(RegistrarVenta(lolla, pt, p, prod, cant)) $\equiv$ {<pt, p, prod, cant>} $\cup$ HistorialVenta(lolla)

HistorialVenta(Hackear(lolla, p, prod)) $\equiv$ HistorialVenta(lolla) - DameVentaHack(HistorialVenta(lolla),p,prod)

DameVentaHack(conjV,p,prod) $\equiv$ **if** VentaHackeable?(DameUno(conjV),p,pr) **then**
DameUno(conjV)
else
DameVentaHack(SinUno(conjV),p,prod)
fi

VentaHackeable?(v,p,prod) $\equiv$ **if** v.p = p $\wedge$ v.prod = prod **then**
if $\neg$TieneDesc?(Descuento(v.puestoC), v.prod, v.cant) **then**
True
else
False
fi
else
False
fi

CalcularGasto(v) $\equiv$ **if** TieneDesc?(Descuento(v.puestoC), v.prod, v.cant) **then**
AplicarDescuento(DevolverPrecio(menu(v.puestoC),v.prod) x v.cant,
DevolverDesc(Descuento(v.puestoC),v.prod, v.cant)
else
DevolverPrecio(menu(v.puestoC),v.prod) x v.cant
fi

GastoPersona(conjV,p) $\equiv$ **if** $\emptyset$?(conjV) **then**
0
else
if DameUno(conjV).p = p **then**
CalcularGasto(DameUno(conjV)) + GastoPersona(SinUno(conjV),p)
else
GastoPersona(SinUno(conjV),p)
fi
fi

BuscarPersona1(ConjV, ConjP) ≡ **if** ∅?(SinUno(ConjP)) **then**
DameUno(ConjP)
else
if GastoPersona(ConjV, DameUno(conjP)) >
GastoPersona(ConjV, BuscarPersona1(conjV, SinUno(ConjP)))
then
DameUno(ConjP)
else
BuscarPersona1(ConjV, SinUno(ConjP))
fi
fi

Persona1(lolla) ≡ BuscarPersona1(HistorialVentas(lolla), Personas(lolla))

Fin TAD