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.co
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 Lollapatuza

TAD Lollapatuza

géneros lollapatuza

igualdad observacional

$$(\forall lolla, lolla' : \text{lollapatuza}) \left(lolla =_{\text{obs}} lolla' \iff \begin{pmatrix} \text{Puestos(lolla)} =_{\text{obs}} \text{Puestos(lolla')} \wedge \\ \text{PesonasHabilitadas(lolla)} \quad =_{\text{obs}} \quad \text{Peso-} \\ \text{nasHabilitadas(lolla')} \end{pmatrix} \right)$$

exporta DevolverPersonal, generadores, PersonasHabilitadas

usa Conjunto, PuestoDeComida, Secuencias, Personas

observadores básicos
Puestos : lollapatuza $\longrightarrow$ conj(puestoDeComida)
PersonasHabilitadas : lollapatuza $\longrightarrow$ conj(Persona)

generadores
CrearLolla : conj(puestoDeComida) $conjPC$ $\longrightarrow$ lollapatuza

$$\left\{ \begin{array}{l} (\forall PC1, PC2\text{: puestoDeComida})(((\text{PC1,PC2}) \in \text{ConjPC}) \rightarrow (\text{ID(PC1)=ID(PC2)} \iff \text{PC1=PC2}) \\ \wedge \\ (\forall PC1, PC2\text{:puestoDeComida})(((\text{PC1,PC2}) \in \text{ConjPC}) \Rightarrow_{\text{L}} (\forall p\text{:Producto}) \\ ((\text{SeEncuentraEnMenu?(InformeMenu(PC1),p)} \wedge \text{SeEncuentraEnMenu?(PC2,p)}) \Rightarrow_{\text{L}} \\ (\text{DevolverPrecio(InformeMenu(PC1,p))} = \text{DevolverPrecio(InformeMenu(PC2,p))}))) \end{array} \right\}$$

RegistrarVenta : lollapatuza $lolla$ $\times$ id nat $\times$ persona p $\times$ producto $prod$ $\times$ nat n $\longrightarrow$ lollapatuza

$$\left\{ \begin{array}{l} \text{p} \in \text{PersonasHabilitadas(lolla)} \wedge (\exists PC : \text{puestoDeComida})(\text{PC} \in \text{Puestos(lolla)} \wedge \text{EstaEnMe-} \\ \text{nu?(InformeMenu(PC),p)} \wedge_{\text{L}} \text{HayStockSuficiente?(InformeStock(PC),prod,n))} \end{array} \right\}$$

LlegaPersona : persona p $\times$ lollapatuza $lolla$ $\longrightarrow$ lollapatuza $\{\text{p} \notin \text{PersonasHabilitadas(lolla)}\}$
Hackear : lollapatuza $lolla$ $\times$ persona p $\times$ producto $prod$ $\longrightarrow$ lollapatuza

$$\left\{ \begin{array}{l} \text{p} \in \text{PersonasHabilitadas(lolla)} \wedge (\exists PC : \text{puestoDeComida})(\text{PC} \in \text{Puestos(Lolla)} \wedge \text{PuestoDeCom-} \\ \text{praHackeable?(PC,p,prod)} \end{array} \right\}$$

otras operaciones
DevolverPersonal : lollapatuza $\longrightarrow$ persona $\{\neg\ \emptyset?\ \text{PersonasHabilitadas(lolla)}\}$
GastosTotales : persona $\times$ conj(PuestosComida) $\longrightarrow$ nat
MaximoGastador : conj(Persona) $\times$ conjPC $\longrightarrow$ persona $\{\neg\ \emptyset?\ \text{conj(Persona)}\}$
RegistroDeCompraHackeable? : secu(<prod $\times$ nat>) $\times$ producto $\times$ listaDeDescuentos $\longrightarrow$ bool
PuestoDeCompraHackeable? : puestoDeComida $\times$ Producto $\times$ persona $\longrightarrow$ bool
HackearPuestos : conj(puestoDeComida) $\times$ persona $\times$ producto $\longrightarrow$ conj(puestoDeComida)

axiomas $\forall conjPC$: conj(puestoDeComida), $\forall lolla$: lollapatuza, $\forall id, cant$: nat, $\forall prod$: Producto, $\forall persona$: p, $\forall hist$: secu(<prod,nat>), $\forall conjPersonas$: conj(Personas), $\forall pt$: Puesto
Puestos(CrearLolla(conjPC)) $\equiv$ conjPC
Puestos(RegistrarVenta(lolla,id,p,prod,cant)) $\equiv$ RegistrarVentaPuestos(Puestos(lolla), id, p, prod, cant)
Puestos(LlegaPersona(lolla, p)) $\equiv$ Puestos(lolla)
Puestos(Hackear(lolla, p, prod)) $\equiv$ HackearPuestos(Puestos(lolla), p, prod)
PersonasHabilitadas(CrearLolla(conjPC)) $\equiv$ $\emptyset$
PersonasHabilitadas(LlegaPersona(lolla, p)) $\equiv$ Ag(p, PersonasHabilitadas(lolla)
PersonasHabilitadas(RegistrarVenta(lolla, id, persona, prod, cant)) $\equiv$ PersonasHabilitadas(lolla)
PersonasHabilitadas(Hackear(lolla, p, prod)) $\equiv$ PersonasHabilitadas(lolla)
PuestoDeCompraHackeable?(pc,prod,p) $\equiv$ RegistroDeCompraHackeable(HistorialDeCompras(pc, p), prod, informeDescuento(pc)

RegistrarVentaPuestos(conjPC, id, p, prod, cant) ≡ **if** ID(DameUno(conjPC)) = id **then**
Ag(RegistrarVenta(DameUno(conjPC),p,prod,cant),
SinUno(conjPC)
else
Ag(DameUno(conjPC),
RegistrarVentaPuestos(SinUno(ConjPC),id,p,prod,cant))
fi

HackearPuestos(conjPC,p,prod) ≡ **if** ∅?(conjPC) **then**
∅
else
if PuestoDeCompraHackeable?(DameUno(ConjPC),prod,p) **then**
Ag(CancelarVenta(DameUno(conjPC),p,prod),SinUno(conjPC))
else
Ag(DameUno(conjPC),HackearPuestos(SinUno(conjPC),p,prod))
fi
fi

RegistroDeCompraHackeable?(hist, prod,d) ≡ **if** Vacia?(hist) **then**
False
else
if prim(hist)[0]=prod **then**
if TieneDescuento?(d,prod,prim(hist)[1]) **then**
RegistroDeCompraHackeable?(fin(hist, prod,d))
else
True
fi
else
RegistroDeCompraHackeable?(fin(hist, prod,d))
fi
fi

MaximoGastador(conjPersonas,conjPC) ≡ **if** ∅?(SinUno(conjPersona)) **then**
DameUno(conjPersonas)
else
if GastosTotales(DameUno(conjPersonas), conjPc) ≥ Gastos-
Totales(MaximoGastadores(SinUno(conjPersonas),conjPc),conjPc)
then
DameUno(conjPersonas)
else
MaximoGastadores(SinUno(conjPersonas))
fi
fi

DevolverPersonal(lolla) ≡ MaximoGastador(personasHabilitadas(lolla), puestos(lolla))

GastosTotales(p,conjPC) ≡ **if** ∅?(conjPC) **then**
0
else
PersonaGasto(DameUno(conjPC), p) + GastosTotales(p, SinUno(conjPC))
fi

Fin TAD

2. TAD Puesto_de_comida

TAD Puesto De Comida

igualdad observacional

$$(\forall pt, pt' : \text{puesto}) \left(pt =_{\text{obs}} pt' \iff \begin{pmatrix} (\text{InformeStock(pt)}=_{\text{obs}} \text{InformeStock(pt')} \wedge \\ \text{InformeMenu(pt)}=_{\text{obs}} \text{InformeMenu(pt')} \wedge \\ \text{InformeDescuento(pt)}=_{\text{obs}} \\ \text{InformeDescuento(pt')} \wedge \\ \text{ID(pt)}=_{\text{obs}} \text{ID(pt')} \wedge_{\text{L}} \\ (\forall p\text{: persona})(\text{HistorialDeCompra(pt,p)}=_{\text{obs}} \text{ Historial-} \\ \text{DeCompra(pt',p)})) \end{pmatrix} \right)$$

géneros puesto

exporta Generadores, PersonaGasto, Observadores Básico

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

observadores básicos

InformeStock : puesto $\longrightarrow$ stock
InformeMenu : puesto $\longrightarrow$ menu
InformeDescuento : puesto $\longrightarrow$ listadeDescuentos
HistorialDeCompra : puesto $\times$ persona $\longrightarrow$ sec(Producto, cantidad)
ID : puesto $\longrightarrow$ nat

generadores

NuevoPuesto : stock s $\times$ menu m $\times$ descuentos d $\times$ nat id $\longrightarrow$ puesto
{$(\forall pr$: producto)(EstaEnMenu?(m,pr) $\Rightarrow_{\text{L}}$ EstaEnListadoStock?(m,pr)}
RealizarVenta : puesto pt $\times$ persona p $\times$ producto pr $\times$ nat n $\longrightarrow$ puesto
{EstaEnMenu?(InformeMenu(pt),p) $\wedge_{\text{L}}$ HayStockSuficiente?(InformeStock(pt), p, n)}
CancelarVenta : puesto pt $\times$ persona p $\times$ producto pr $\longrightarrow$ puesto
$\left\{ \begin{array}{l} (\exists n\text{:N})(0 \leq \text{n} < \text{Longitud(HistorialDeCompra(pt,p))} \quad \wedge_{\text{L}} \text{HistorialDeCompra(pt,p)[n][0]=pr} \\ \wedge \neg \text{TieneDescuento?(InformeDescuento(pt),pr,HistorialDeCompra(pt,p)[n][1]))} \end{array} \right\}$

otras operaciones

EliminarDeTupla : secu(<Producto $\times$ nat>) $\longrightarrow$ secu(<Producto, nat>)
CantidadDeProductoAAgregar : secu(<Producto $\times$ nat>) $\times$ Producto $\longrightarrow$ nat
div : nat p $\times$ nat k $\longrightarrow$ nat {0 < k}
aplicarDescuento : nat p $\times$ nat d $\longrightarrow$ nat {d < 100}
sumarGastosHistorial : secu(<Producto $\times$ nat>) $\times$ InformeMenu $\times$ InformeDescuento $\longrightarrow$ nat
PersonaGasto : puestoDeComida $\times$ persona $\longrightarrow$ nat

axiomas $\forall s$: stock, $\forall m$: menu, $\forall d$: listaDeDescuentos, $\forall id, n, k$: nat, $\forall prod$: Producto, $\forall pt$: puesto, $\forall p$,p$_1$,p_2: $persona$,$\forall s$: secu(<Producto, nat>)

InformeStock(NuevoPuesto(s,m,d,id)) $\equiv$ s
InformeStock(RealizarVenta(pt,p,prod,n)) $\equiv$ SacarStock(InformeStock(pt),prod,n)
InformeStock(CancelarCompra(pt,p,prod)) $\equiv$ AgregarStock(InformeStock(pt),prod,
CantidadDeProductoAAgregar(HistorialDeCompra(pt,p),prod,
InformeDescuento(pt))
InformeMenu(NuevoPuesto(s,m,d,id)) $\equiv$ m
InformeMenu(RealizarVenta(pt,p,prod,n)) $\equiv$ InformeMenu(pt)
InformeMenu(CancelarCompra(pt,p,prod)) $\equiv$ InformeMenu(pt)
InformeDescuento(NuevoPuesto(s,m,d,id)) $\equiv$ d
InformeDescuento(RealizarVenta(pt,p,prod,n)) $\equiv$ InformeDescuento(pt)
InformeDescuento(CancelarCompra(pt,p,prod)) $\equiv$ InformeDescuento(pt)
ID(NuevoPuesto(s,m,d,id)) $\equiv$ id
ID(NuevoPuesto(RealizarVenta(pt,p,prod,n)) $\equiv$ ID(pt)
ID(NuevoPuesto(CancelarCompra(pt,p,prod)) $\equiv$ ID(pt)
HistorialDeCompra(NuevoPuesto(s,m,d,id),p) $\equiv$ <>
HistorialDeCompra(RealizarVenta(pt,p_1,prod,n),p_2) $\equiv$ **if** p_1=p_2 **then**
<prod,n> • HistorialDeCompra(pt,p_1)
else
HistorialDeCompra(pt,p_2)
fi

HistorialDeCompra(CancelarCompra(pt,p_1,prod),p_2) ≡ **if** p_1=p_2 **then**
EliminarDeTupla(HistorialDeCompra(pt,p_1),prod,
InformeDescuento(pt))
else
HistorialDeCompra(pt,p_2)
fi

EliminarDeTupla(s,prod,d) ≡ **if** vacia?(s) **then**
<>
else
if prim(s)[0]=prod **then**
if ¬TieneDescuento?(d,prim(s)[0],prim(s)[1]) **then**
fin(s)
else
prim(s) • EliminarDeTupla(fin(s),prod,d)
fi
else
prim(s) • EliminarDeTupla(fin(s),prod,d)
fi
fi

CantidadDeProductoAAgregar(s,prod,d) ≡ **if** vacia?(s) **then**
0
else
if prim(s)[0]=prod **then**
if ¬TieneDescuento?(d,prim(s)[0],prim(s)[1]) **then**
prim(s)[1]
else
CantidadDeProductoAAgregar(fin(s),prod,d)
fi
else
CantidadDeProductoAAgregar(fin(s),prod,d)
fi
fi

div(n,k) ≡ **if** n<k **then** 0 **else** 1+div(n-k, k) **fi**

aplicarDescuento(p,d) ≡ div(p x (100-d), 100)

sumarGastosHistorial(s,d,m) ≡ **if** vacia?(s) **then**
0
else
if TieneDescuento?(d,prim(s)[0],prim(s)[1]) **then**
AplicarDescuento(DevolverPrecio(m,prim(s)[0]) x prim(s)[1]),
DevolverDescuento(d,prim(s)[0],prim[1]) +
sumarGastosHistorial(fin(s),d,m)
else
DevolverPrecio(m,prim(s)[0]) x prim(s)[1] +
sumarGastosHistorial(fin(s),d,m)
fi
fi

PersonaGasto(pc,p) ≡ sumarGastosHistorial(HistorialDeCompra(pc,p),InformeMenu(pc),InformeDescuento(pc))

Fin TAD

3. TAD Menu

TAD Menu

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

géneros menu

exporta Generadores, EstaEnMenu?, DevolverPrecio

usa Bool, Nat, Diccionario, Producto

observadores básicos

DevolverDicc : 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

DevolverDicc(NuevoMenu(m)) $\equiv$ m

EstaEnMenu?(menu,prod) $\equiv$ def?(prod,DevolverDicc(menu))

DevolverPrecio(menu,prod) $\equiv$ obtener(prod,DevolverDicc(menu))

Fin TAD

4. TAD Stock

TAD Stock

igualdad observacional

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

géneros stock

exporta Generadores, EstaEnListadoStock?, DevolverStock, HayStockSuficiente?

usa Bool, Nat, Diccionario, Producto

observadores básicos

DevolverDicc : 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, $\forall n, cant$: nat

DevolverDicc(s) $\equiv$ s

DevolverDicc(AgregarStock(s, prod, n) $\equiv$ definir(prod, obtener(prod, DevolverDicc(s)) + n, borrar(prod, DevolverDic(s))

DevolverDicc(SacarStock(s, prod, n) $\equiv$ definir(prod, obtener(prod, DevolverDic(s)) - n, borrar(prod, DevolverDic(s))

EstaEnListadoStock?(stock,prod) $\equiv$ def?(prod, DevolverDicc(stock)

DevolverStock(stock,prod) $\equiv$ obtener(prod, DevolverDic(stock))

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

Fin TAD

5. TAD Lista De Descuentos

TAD Lista De Descuentos

igualdad observacional

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

géneros listadeDescuentos

exporta Generadores, TieneDescuento?, DevolverDescuento

usa Bool, Nat, Diccionario, Producto, Tupla

observadores básicos

DevolverDicc : listadeDescuentos $\longrightarrow$ dicc(tupla(Producto, nat), nat)

generadores

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

otras operaciones

ClavesDesc : listaDeDescuentos $\longrightarrow$ secu(<Producto, nat>)
EncontrarCantiDescuento : secu(<Producto $\times$ nat>) $\times$ Producto $\times$ nat $\longrightarrow$ nat
TieneDescuento? : listaDeDescuentos $\times$ Producto $\times$ nat $\longrightarrow$ bool
DevolverDescuento : listaDeDescuentos ls $\times$ Producto p $\times$ nat n $\longrightarrow$ nat {TieneDescuento?(ls, p, n)}

axiomas ∀*ls*: listaDeDescuentos, ∀*prod*: Producto, ∀*cant*: nat, ∀*secProd*: secu(<Producto, nat>)

DevolverDicc(NuevaListaDeDescuentos(ls)) ≡ ls
ClavesDesc(ls) ≡ claves(DevolverDicc(ls))
EncontrarCantiDescuento(secProd, prod, cant) ≡ **if** *vacia?*(*secProd*) **then**
0
else
if Prim(secProd)[0] = prod)
then
if Prim(secProd)[1] ≤ cant
then
max (prim(secProd)[1],
EncontrarCantiDescuento(fin(secProd),prod,cant))
else
EncontrarCantiDescuento(fin(secProd),prod,cant)
fi
else
EncontrarCantiDescuento(fin(secProd), prod, cant)
fi
fi
TieneDescuento?(listaDesc, prod, cant) ≡ ¬(EncontrarCantiDescuento(ClavesDesc(listaDesc),prod,cant)=0?)
DevolverDescuento(listaDesc, prod, cant) ≡ obtener(⟨prod,EncontrarCantiDescuento(ClavesDesc(listaDesc),
prod,cant)⟩, DevolverDicc(listaDesc))

Fin TAD