Algoritmos y Estructuras de Datos II

Trabajo Práctico 1

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

Lollapatuza

Integrante	LU	Correo electrónico
Agustin Fernandez Aragon	998/21	f.a.agustin@gmail.com
Bruno Muschietti	924/21	brunomuschi@gmail.com
Felipe Saidón	1436/21	felipesaidon@gmail.com
Matias Daniel Diaz Sarmiento	704/19	mdds.2017@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Renombres de TADs

TAD Comida es String

TAD Consumo es Tupla<Comida,Puesto,Cantidad>

TAD Descuento es Tupla<Comida,Cantidad,Nat>

TAD Cantidad es Nat

TAD Menu es Diccionario(Comida,Nat)

TAD stock es Diccionario(comida,cantidad)

2. Desarrollo

2.1. TAD Puestos de Comida

TAD Puesto

igualdad observacional

$$(\forall pc1, pc2 : \text{puesto}) \left(pc1 =_{\text{obs}} pc2 \iff \begin{pmatrix} \text{ID(pc1)}=_{\text{obs}}\text{ID(pc2)} \wedge \\ \text{stock(pc1)}=_{\text{obs}}\text{stock(pc2)} \wedge \\ \text{menu(pc1)}=_{\text{obs}}\text{menu(pc2)} \wedge \\ \text{descuento(pc1)}=_{\text{obs}}\text{descuento(pc2)} \end{pmatrix} \right)$$

géneros puesto

exporta puesto, generadores, observadores, operaciones

usa string, diccionario(clave,significado), nat, conjunto(tupla$< \pi_1, \pi_2, \pi_3 >$), festival, persona

generadores

crearPuesto : string id × stock s × menu m × conj(descuento) d $\longrightarrow$ puesto
{claves(m) = claves(s) ∧ noHayDescIguales(d)}

venta : puesto pc × comida c × cantidad $cant$ $\longrightarrow$ puesto
{c ∈ claves(stock(pc)) $\wedge_L$ cant≤obtener(c,stock(pc))}

observadores básicos

ID : puesto $\longrightarrow$ nat
stock : puesto $\longrightarrow$ stock
menu : puesto $\longrightarrow$ menu
descuentos : puesto $\longrightarrow$ conj(descuento)

otras operaciones

noHayDescIguales : conj(descuento) d $\longrightarrow$ bool
enEsteNoHayIguales : conj(descuento) d × descuento t $\longrightarrow$ bool

axiomas

ID(crearPuesto(id, s, m, d)) ≡ id
ID(venta(p,com)) ≡ ID(p)
stock(crearPuesto(id,s, m, d)) ≡ s
stock(venta(pc,com) ≡ definir(com, obtener(com,stock(pc)) − 1, borrar(com,stock(pc)))
menu(crearPuesto(id, s, m, d)) ≡ m
menu(venta(pc,com)) ≡ menu(pc)
descuentos(crearPuesto(id, s, m, d)) ≡ d
descuentos(venta(pc,com)) ≡ descuentos(pc)
noHayDescIguales(d) ≡ **if** esvacio?(d) **then**
true
else
if enEsteNoHayIguales(sinUno(d),dameUno(d)) **then**
true ∧ noHayDescIguales(sinUno(d))
else
false
fi
fi

enEsteNohayIguales(d,t) ≡ **if** esvacio?(d) **then**
true
else
if π_1(dameUno(d)) = π_1(t) ∧ π_2(dameUno(d)) =π_2(t) **then**
false
else
enEsteNoHayIguales(sinUno(d),t)
fi
fi

Fin TAD

2.2. TAD Persona

TAD PERSONA

igualdad observacional

$$(\forall p_1, p_2 : \text{persona}) \left(p_1 =_{\text{obs}} p_2 \iff \begin{pmatrix} \text{ID(p1)} =_{\text{obs}} \text{ID(p2)} \wedge \\ \text{consumos(p1)} =_{\text{obs}} \text{consumos(p2)} \end{pmatrix} \right)$$

géneros persona

exporta persona, generadores, observadores, operaciones

usa string, multiconjunto(tupla< π_1, π_2, π_3 >), nat, bool, conjunto(tupla< π_1, π_2, π_3 >), festival, puesto

generadores
nueva : string *id* × multiconjunto(consumo) ⟶ persona
comprar : persona *p* × comida *c* × puesto *pc* × cantidad *cant* ⟶ persona
{c ∈ claves(stock(pc)) $\wedge_L$ cant ≤ obtener(c,stock(pc))}

observadores básicos
IDEN : persona ⟶ string
consumos : persona ⟶ multiconjunto(consumo)

otras operaciones
cuantoGasto : persona ⟶ nat
elDescuento : comida *com* × cantidad *cant* × conjunto(descuento) *descuentos* ⟶ nat
{(∃ desc : descuento)(desc ∈ descuentos ∧ π_1(desc)=com ∧ π_2(desc) ≤ cant)}
hayUnDescMayor : cantidad *c* × cantidad *desc* × conj(descuento) × comida ⟶ bool

axiomas
IDEN(nueva(id, cons)) ≡ *id*
IDEN(comprar(p,c,l, cant)) ≡ IDEN(p)
consumos(nueva(id, cons)) ≡ cons

consumos(comprar(p,com,pc, cant)) ≡ Ag(tupla< *com*, *pc*, *cant* >,consumos(p))

cuantoGasto(consumos(p_1)) ≡ **if** ∅?(consumos(*p*)) **then**
0
else
if
huboDescuento?(π_1(dameUno(consumos(p_1))),
descuentos(π_2(dameUno(consumos(p_1))), π_3(dameUno(consumos(p_1)))))
then
aplicarDescuento(obtener(com,menu(dameUno(π_2(consumos(p_1)))))) ∗
π_3(dameUno(consumos(p_1))),
elDescuento(π_1(dameUno(consumos(p_1)))),π_3(dameUno(consumos(p_1))),
descuentos(π_2(dameUno(consumos(p_1))))
+ cuantoGasto(sinUno(consumos(p_1)))
else
obtener(com,menu(dameUno(π_2(consumos(p_1))))) ∗
π_3(dameUno(consumos(p_1))) + cuantoGasto(sinUno(consumos(p_1)))
fi
fi

elDescuento(com_1,$cant_1$,desc)) $\equiv$ **if** $com_1 = \pi_1$(dameUno(desc)) $\wedge$
π_2(dameUno(desc)) $\leq$cant$_1$ $\wedge$
$\neg(hayUnDescMayor$(cant$_1$,π_2(dameUno(desc)),sinUno(desc),com_1))
then
π_3(dameUno(desc))
else
elDescuento(com_1,$cant_1$,sinUno(desc))
fi

hayUnDescMayor(cant1,cant2,desc,com1) $\equiv$ **if** $\emptyset?(desc)$
then
False
else
if $com_1 = \pi_1$(dameUno(desc)) $\wedge$
π_2(dameUno(desc)) $\leq cant_1$ $\wedge$
cant$_2$ $< \pi_2$(dameUno(desc))
then
True
else
$\neg HayUndescMayor$(cant$_1$,$cant_2$,sinUno(desc),com_1)
fi
fi

Fin TAD

2.3. TAD Festival

TAD Festival

igualdad observacional

$$(\forall f1, f2 : \text{festival}) \left(f1 =_{\text{obs}} f2 \iff \begin{pmatrix} \text{personas(f1)} =_{\text{obs}} \text{personas(f2)} \wedge \\ \text{puestos(f1)} =_{\text{obs}} \text{puestos(f2)} \end{pmatrix} \right)$$

géneros festival

exporta festival, generadores, observadores, operaciones

usa string, multiconjunto(tupla$< \pi_1, \pi_2, \pi_3 >$), nat, bool, conjunto(α), persona, puesto

generadores

abrir : conj(personas) $\times$ conj(puestos) $\longrightarrow$ festival
hacker : festival $\times$ persona $\times$ comida $\longrightarrow$ festival
compra/venta : festival f $\times$ persona p $\times$ puesto pc $\times$ comida com $\times$ cantidad $cant$ $\longrightarrow$ festival
{p $\in$ personas(f) $\wedge$ pc $\in$ puestos(f) $\wedge$ com $\in$ claves(stock(pc)) $\wedge$ cant $\leq$ obtener(com,stock(pc))}

observadores básicos

personas : festival $\longrightarrow$ conj(personas)
puestos : festival $\longrightarrow$ conj(puestos)

otras operaciones

mayorGastador : festival f $\longrightarrow$ persona {$\neg$esvacio?(personas(f))}
mayor : conj(personas) $\longrightarrow$ nat
huboDescuento : comida $\times$ conjunto(descuento) $\times$ cantidad $\longrightarrow$ bool
puedeHackear : multiconj(consumo) $\times$ comida $\longrightarrow$ bool
consumoHackeado : multiconj(consumo) $cons$ $\times$ comida com $\longrightarrow$ consumo {puedeHackear($cons$,com)}
personaHackeada : persona $\times$ multiconj(consumo) $\times$ consumo $\longrightarrow$ persona
consYaHackeados : multiconj(consumo) $\times$ consumo $\longrightarrow$ multiconj(consumo)
puestoHackeado : puesto pc $\times$ comida c $\longrightarrow$ puesto {c $\in$ claves(stock(pc))}
StockModificado : puesto pc $\times$ comida c $\longrightarrow$ sotck {c $\in$ claves(stock(pc))}
compró : conj(personas) $pers$ $\times$ persona p $\times$ puesto pc $\times$ comida c $\times$ cant $cant$ $\longrightarrow$ persona
{p $\in$ pers $\wedge$ c $\in$ claves(stock(pc)) $\wedge_L$ cant $\leq$ obtener(c,stock(pc))}

vendió : conj(puestos) *pcs* × puesto *pc* × comida *c* × cant *cant* ⟶ puesto
{pc ∈ pcs ∧ c ∈ claves(stock(pc)) ∧$_L$ cant ≤ obtener(c,stock(pc))}

axiomas

personas(abrir(conj(persona),conj(puestos)) ≡ conj(persona)

personas(hacker(f,p,com)) ≡ **if** p ∉ personas(f) **then**
personas(f)
else
if dameUno(personas(f)) = p **then**
if puedeHackear(consumos(p),com) **then**
Ag(personaHackeada(p,consumos(p),consumoHackeado(consumos(p),com)),
personas(f) - {p})
else
personas(f)
fi
else
Ag(dameUno(personas(f)),
personas(hacker(abrir(sinUno(personas(f)),puestos(f)),p,com)))
fi
fi

personas(compra/venta(f,pers,pc,com,cant)) ≡ Ag(compró(personas(f),pers,pc,com,cant),personas(f) - {pers})

puestos(abrir(conj(persona),conj(puestos))) ≡ conj(puestos)

puestos(hacker(f,p,com)) ≡ **if** π_2(consumoHackeado(consumos(p),com)) ∉ puestos(f) **then**
puestos(f)
else
if puedeHackear(f,p,com) ∧ π_2(consumoHackeado(consumos(p),com)) = dameUno(puestos(f)) **then**
Ag(puestoHackeado(π_2(consumoHackeado(consumos(p),com)),com), puestos(f) - p)
else
Ag(dameUno(puestos(f)),
puestos(hacker(abrir(personas(f),sinUno(puestos(f)),p,com)))))
fi
fi

puestos(compra/venta(f,pers,pc,com,cant)) ≡ Ag(vendió(puestos(f),pc,com,cant),puestos(f)-{pc})

mayorGastador(f) ≡ **if** cuantoGasto(dameUno(personas(f))) = mayor(personas(f)) **then**
dameUno(personas(f))
else
mayorGastador(abrir(sinUno(personas(f)),puestos(f)))
fi

mayor(conj(personas)) ≡ **if** esvacio?(conj(persona)) **then**
0
else
max(cuantoGasto(dameUno(conj(persona))), mayor(sinUno(conj(persona))))
fi

huboDescuento(com,desc,cant) ≡ **if** π_1(dameUno(desc)) = com ∧ π_2(dameUno(desc)) ≤ cant **then**
true
else
false ∨ huboDescuento(com,sinUno(desc),cant)
fi

puedeHackear(cons,com) ≡ **if** π_1(dameUno(cons)) = com ∧
¬huboDescuento(com,descuentos($\pi_2$dameUno(cons)),π_3(dameUno(cons))) **then**
True
else
consumoHackeado(sinUno(cons),com) ∨ False
fi

consumoHackeado(cons,com) ≡ **if** π_1(dameUno(cons)) = com ∧
¬ huboDescuento(com,descuentos(π_2(dameUno(cons))),π_3(dameUno(cons)))
then
dameUno(cons)
else
consumoHackeado(sinUno(cons,com))
fi

personaHackeada(p,cons,consHack) ≡ nueva(IDEN(p),consYaHackeados(cons,consHack))

consYaHackeados(cons,consHack) ≡ Ag(tupla<π_1(consHack),π_2(consHack),π_3(consHack)−1>,cons−{consHack})

puestoHackeado(pcHack,com) ≡ crearPuesto(id(pcHack),StockModificado(pcHack,com),menu(pcHack),descuento(pcHa

StockModificado(pcHack,com) ≡ definir(com,obtener(com,stock(pcHack))-1,stock(pcHack))

compró(pers,p,pc,com,cant) ≡ **if** dameUno(pers) = p **then**
comprar(p,com,pc,cant)
else
compró(sinUno(pers),p,pc,com,cant)
fi

vendió(pcs,pc,com,cant) ≡ **if** dameUno(pcs) = pc **then**
venta(pc,com,cant)
else
vendió(sinUno(pcs),pc,com,cant)
fi

Fin TAD

3. Comentarios

3.1. TAD Puestos

Operaciones:

noHayDescIguales: Esta función verifica que en un conjunto de descuentos no existan dos promociones para la misma comida y la misma cantidad de compra. Esto lo hace tomando de a un descuentos y comprándolo con todos los demás. Únicamente la usamos para la restricción de crearPuesto.

3.2. TAD Persona

Generadores:

nueva: Decidimos que nueva persona tome también los consumos (puede ser vacío) para poder utilizarla en la axiomatización de hacker

Operaciones:

cuantoGasto: Calcula el gasto total de una persona Iterando a través de sus consumos, si hubo descuento llama a la función aplicar descuento con el valor de la compra (cantidad * precio) y el porcentaje a descontar. Si no hubo descuento directamente calcula el valor de la compra. Por último, el precio obtenido lo suma a la recursión del resto de consumos

elDescuento:A partir de una comida, una cantidad comprada y las promociones de un puesto de comida devuelve el mayor descuento aplicable a la compra. Esto lo obtiene verificando que no exista otro descuento para el mismo producto y que la cantidad requerida sea mayor al del primero y, a su vez, menor a la comprada (función hayUnDescMayor).

3.3. TAD Festival

Operaciones:

mayorGastado: Devuelve la persona la cual su cantidad gastada coincida con la función mayor. Es decir, retorna la persona que mas gasto en el festival.
mayor: De un conjunto de personas devuelve el natural que corresponde al mayor gasto

huboDescuento: Dado una comida, una cantidad y las promociones de un puesto, devolverá true si existe un descuento para ese tipo de compra (comida y cantidad)

puedeHackear: Recibe los consumos de una persona y una comida y devuelve un booleano que dependerá de si existe alguna compra de esa comida en particular y que haya sido efectuada sin descuento

consumoHackeado: Igual que puede hackear pero en vez de retornar un booleano devuelve la compra que cumple con los requisitos de hacker

personaHackeada: Crea la nueva persona resultante del hackeo, con el mismo id y los consumos modificados

consYaHackeados: Devuelve los consumos de la persona hackeada con la unica modificacion en la compra que cumple con las condiciones del hackeo.

puestoHackeado: Crea un nuevo puesto a partir del puesto donde se realizo la compra a hackear con los mismos datos exceptuando la reposición del stock.

StockModificado: Dado el puesto a hackear y la comida, redefine el stock devolviendo a su lugar el item previamente comprado

Compró: A partir de un conjunto de personas, una persona y un consumo, busca la persona dentro del conjunto y una vez encontrada (lo hará dado a la restricción de compra/venta), le realiza la compra dada.

Vendió: A partir de un conjunto de puestos, un puesto y un consumo, busca el local dentro del conjunto y una vez encontrado, le realiza la venta dada.