1. Módulo Ciudad Robotica

Interfaz

se explica con: SECUENCIA(α), ITERADOR BIDIRECCIONAL(α).

géneros: `ciudad`, `itLista(`α`)`.

usa:

Operaciones básicas de ciudad

CREAR(**in** m: `mapa`) $\rightarrow$ res : `ciudad`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ crear(m)}
Complejidad: $\Theta(1)$
Descripción: genera una nueva Ciudad.
Aliasing: lo ideal seria no copiar ese mapa no?

ENTRAR(**in** ts: `conj(tag)`, **in** e: `estacion`, **in/out** c: `ciudad`)
Pre $\equiv$ {$c =_{\text{obs}} c_0 \wedge e \in$ estaciones(c)}
Post $\equiv$ {$c =_{\text{obs}}$ entrar(ts, e, c_0)}
Complejidad: $\Theta(copy(a))$..

MOVER(**in** u: `rur`, **in** e: `estacion`, **in/out** c: `ciudad`)
Pre $\equiv$ {$c =_{\text{obs}} c_0 \wedge e \in$ estaciones(c)$\wedge \in$ robots(c)}
Post $\equiv$ {$c =_{\text{obs}}$ mover(u, e, c_0)}
Complejidad: $\Theta(copy(a))$..

INSPECCION(**in** e: `estacion`, **in/out** c: `ciudad`)
Pre $\equiv$ {$c =_{\text{obs}} c_0 \wedge e \in$ estaciones(c)}
Post $\equiv$ {$c =_{\text{obs}}$ inspeccion(e, c_0)}
Complejidad: $\Theta(copy(a))$..

PROXIMORUR(**in** c: `ciudad`) $\rightarrow$ res : `rur`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ proximoRUR(c)}
Complejidad: $O(1)$..

MAPA(**in** c: `ciudad`) $\rightarrow$ res : `mapa`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ mapa(c)}
Complejidad: $\Theta(copy(a))$..

ROBOTS(**in** c: `ciudad`) $\rightarrow$ res : `it(conj(rur))`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ CrearIt(robots(c))}
Complejidad: $O(1)$..

ESTACION(**in** u: `rur`, **in** c: `ciudad`) $\rightarrow$ res : `estacion`
Pre $\equiv$ {$u \in$ robots(c)}
Post $\equiv$ {$res =_{\text{obs}}$ estacion(u, c)}
Complejidad: $O(1)$..

TAGS(**in** u: `rur`, **in** c: `ciudad`) $\rightarrow$ res : `conj(tag)`
Pre $\equiv$ {$u \in$ robots(c)}
Post $\equiv$ {$c =_{\text{obs}}$ inspeccion(e, c_0)}
Complejidad: $O(1)$..

#INFRACCIONES(**in** u: `rur`, **in** c: `ciudad`) $\rightarrow$ res : `nat`
Pre $\equiv$ {$u \in$ robots(c)}

Post $\equiv \{c =_{\text{obs}} \text{inspeccion}(e, c_0)\}$
Complejidad: $\Theta(copy(a))$..

ESTACIONES(**in** c: `ciudad`) $\rightarrow$ *res* : `it(conj(estaciones))`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{CrearIt}(\text{estaciones}(c))\}$
Complejidad: $O(1)$..

Representación

Representación de la ciudad

`ciudad` **se representa con** `str`

donde `str` es `tupla`(*robRUR*: `DiccArreglo(rur, datosRobot)`,
robEstacion: `DiccTrie(estacion, colaPrio(rur))`,
mapa: `DiccTrie(estacion, DiccTrie(estacion, Restriccion))`)
donde `datosRobot` es `tupla`(*presente?*: `bool`,
est: `estacion`,
infr: `nat`,
tags: `conjTrie(tags)`,
permisos: `DiccTrie(estacion, DiccTrie(estacion, bool)`,
itEst: `it(colaPrio(rur))`)

Rep : lst $\longrightarrow$ bool
Rep(l) $\equiv$ true $\iff$ (l.primero = NULL) = (l.longitud = 0) $\wedge_{\text{L}}$ (l.longitud $\neq 0 \Rightarrow_{\text{L}}$
Nodo(l, l.longitud) = l.primero $\wedge$
($\forall i$: nat)(Nodo(l,i)$\rightarrow$siguiente = Nodo(l,$i+1$)$\rightarrow$anterior) $\wedge$
($\forall i$: nat)($1 \leq i < l$.longitud $\Rightarrow$ Nodo(l,i) $\neq$ l.primero)

Nodo : lst l $\times$ nat $\longrightarrow$ puntero(nodo) $\{l.\text{primero} \neq \text{NULL}\}$
Nodo(l,i) $\equiv$ **if** $i = 0$ **then** l.primero **else** Nodo(FinLst(l), $i-1$) **fi**

FinLst : lst $\longrightarrow$ lst
FinLst(l) $\equiv$ Lst(l.primero$\rightarrow$siguiente, l.longitud $-$ mín$\{l$.longitud, 1$\}$)

Lst : puntero(nodo) $\times$ nat $\longrightarrow$ lst
Lst(p, n) $\equiv$ $\langle p, n \rangle$

Abs : lst l $\longrightarrow$ secu(α) $\{\text{Rep}(l)\}$
Abs(l) $\equiv$ **if** l.longitud = 0 **then** <> **else** l.primero$\rightarrow$dato $\bullet$ Abs(FinLst(l)) **fi**

Representación del iterador

`itLista(`α`)` **se representa con** `iter`

donde `iter` es `tupla`(*siguiente*: `puntero(nodo)`, *lista*: `puntero(lst)`)

Rep : iter $\longrightarrow$ bool
Rep(it) $\equiv$ true $\iff$ Rep($*(it$.lista)) $\wedge_{\text{L}}$ (it.siguiente = NULL $\vee_{\text{L}}$ ($\exists i$: nat)(Nodo($*it$.lista, i) = it.siguiente)

Abs : iter it $\longrightarrow$ itBi(α) $\{\text{Rep}(it)\}$
Abs(it) $=_{\text{obs}}$ b: itBi(α) | Siguientes(b) = Abs(Sig(it.lista, it.siguiente)) $\wedge$
Anteriores(b) = Abs(Ant(it.lista, it.siguiente))

Sig : puntero(lst) l $\times$ puntero(nodo) p $\longrightarrow$ lst $\{\text{Rep}(\langle l, p \rangle)\}$
Sig(i, p) $\equiv$ Lst(p, $l\rightarrow$longitud $-$ Pos($*l$, p))

Ant : puntero(lst) l $\times$ puntero(nodo) p $\longrightarrow$ lst $\{\text{Rep}(\langle l, p \rangle)\}$
Ant(i, p) $\equiv$ Lst(**if** $p = l\rightarrow$primero **then** NULL **else** $l\rightarrow$primero **fi**, Pos($*l$, p))

Nota: cuando p = NULL, Pos devuelve la longitud de la lista, lo cual está bien, porque significa que el iterador no tiene siguiente.

Pos : lst l × puntero(nodo) p $\longrightarrow$ puntero(nodo) {Rep($\langle l, p \rangle$)}

Pos(l,p) $\equiv$ **if** l.primero = $p \vee l$.longitud = 0 **then** 0 **else** 1 + Pos(FinLst(l), p) **fi**

Algoritmos

2. Modulo Restriccion

Interfaz

se explica con: RESTRICCION

géneros: `restriccion`

usa:

<• >(**in** t: `tag`) → res : `restriccion`
Pre ≡ {true}
Post ≡ {res =$_{\text{obs}}$ <t>}
Complejidad: Θ(1)
Descripción: genera una restriccion de un solo tag.

NOT • (**in** r: `restriccion`) → res : `restriccion`
Pre ≡ {true}
Post ≡ {res =$_{\text{obs}}$ NOT r}
Complejidad: Θ(1)..

• AND • (**in** $r1$: `restriccion`, **in** $r2$: `restriccion`) → res : `restriccion`
Pre ≡ {true}
Post ≡ {res =$_{\text{obs}}$ $r1$ AND $r2$}
Complejidad: Θ(1).. iInaugurarWolfie • OR • (**in** $r1$: `restriccion`, **in** $r2$: `restriccion`) → res : `restriccion`
Pre ≡ {true}
Post ≡ {res =$_{\text{obs}}$ $r1$ OR $r2$}
Complejidad: Θ(1)..

VERIFICA?(**in** ts: `conj(tag)`, **in** r : `restriccion`) → res : `bool`
Pre ≡ {true}
Post ≡ {res =$_{\text{obs}}$ verifica?(ts, r)}
Complejidad: $O(R)$..

Representación

Representacion de la restriccion

`restriccion` **se representa con** `rtr`

donde `rtr` es `tupla`(*raiz*: `puntero(Nodo)`)
donde `Nodo` es `tupla`(*tag*: `tag` ,
tipo: `log` ,
negado?: `bool` ,
izq: `puntero(Nodo)` ,
der: `puntero(Nodo)`)
donde log es enum{AND, OR, CAR}

Rep : rtr $\longrightarrow$ bool
Rep(r) $\equiv$ true $\iff$ noCiclos(r.raiz, $\emptyset$) $\wedge_{\text{L}}$esRestriccion?(r.raiz)
ESRESTRICCION?(**in** n: `Nodo`) → res : `bool`
if n.tipo =$_{\text{obs}}$ CAR
then n.izq =$_{\text{obs}}$ NULL ∧ n.der =$_{\text{obs}}$ NULL

else n.izq $\neq$ NULL $\wedge$ n.der $\neq$ NULL $\wedge_{\text{L}}$
esRestriccion?(n.izq) $\wedge$ esRestriccion?(n.der)
end if

Algoritmos

$<\bullet>$(**in** t: `tag`) $\rightarrow$ res : `restriccion`
puntero(Nodo) n $\leftarrow$ New Nodo
n.tag $\leftarrow$ t
n.tipo $\leftarrow$ CAR
n.negado $\leftarrow$ false
n.izq $\leftarrow$ NULL
n.der $\leftarrow$ NULL
res.raiz $\leftarrow$ n

NOT $\bullet$ (**in**/**out** $r1$: `restriccion`)
r1.negado? $\leftarrow$ true

$\bullet$ AND $\bullet$ (**in** $r1$: `restriccion`, **in** $r2$: `restriccion`) $\rightarrow$ res : `restriccion`
puntero(Nodo) n $\leftarrow$ New Nodo
n.tipo $\leftarrow$ AND
n.negado $\leftarrow$ false
n.izq $\leftarrow$ r1.raiz
n.der $\leftarrow$ r2.raiz
res.raiz $\leftarrow$ n

$\bullet$ OR $\bullet$ (**in** $r1$: `restriccion`, **in** $r2$: `restriccion`) $\rightarrow$ res : `restriccion`
puntero(Nodo) n $\leftarrow$ New Nodo
n.tipo $\leftarrow$ OR
n.negado $\leftarrow$ false
n.izq $\leftarrow$ r1.raiz
n.der $\leftarrow$ r2.raiz
res.raiz $\leftarrow$ n

VERIFICA(**in** r : `restriccion`, **in** ts: `tags`) $\rightarrow$ res : `bool`
res $\leftarrow$ verificaAux(r.raiz, ts)

VERIFICAAUX(**in** n: `Nodo`, **in** ts: `tags`) $\rightarrow$ res : `bool`
bool aux $\leftarrow$ false
if n.tipo = CAR
then aux $\leftarrow$ pertenece?(n.tag, ts)
else if n.tipo = AND
then aux $\leftarrow$ verificaAux(n.izq, ts) $\wedge$ verificaAux(n.der, ts)
else aux $\leftarrow$ verificaAux(n.izq, ts) $\vee$ verificaAux(n.der, ts)
if n.negado? then res $\leftarrow$ $\neg$ aux
else res $\leftarrow$ aux