Recursión avanzada y TADs con comportamiento automático

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

Algoritmos y Estructuras de Datos II

Repasemos...

Una función es **recursiva** si en su definición se llama **a sí misma**.

- **Casos base:** Casos en los que se resuelve sin recursividad.
- **Casos recursivos:** Casos en los que usa recursividad.

Recursión

Extender el tipo Nat con la operación *esPar?* que indica si un número es o no es par.

esPar? : nat $\longrightarrow$ bool { }

esPar?(0) $\equiv$??

esPar?(suc(n)) $\equiv$??

Extender el tipo Nat con la operación *esPar?* que indica si un número es o no es par.

esPar? : nat $\longrightarrow$ bool { }

esPar?(0) $\equiv$ true

esPar?(suc(n)) $\equiv$??

Recursión

Extender el tipo Nat con la operación *esPar?* que indica si un número es o no es par.

$\text{esPar?} : \text{nat} \longrightarrow \text{bool}$ $\{\ \}$

$\text{esPar?}(0) \equiv \text{true}$

$\text{esPar?}(\text{suc}(n)) \equiv \neg\ \text{esPar?}(n)$

Extender el tipo Nat con la operación *esPar?* que indica si un número es o no es par.

esPar? : nat $\longrightarrow$ bool { }

esPar?(0) $\equiv$ true

esPar?(suc(n)) $\equiv$ $\neg$ esPar?(n)

Podemos ahora agregar *esImpar?* de la siguiente manera:

esImpar? : nat $\longrightarrow$ bool { }

esImpar?(0) $\equiv$ false

esImpar?(suc(n)) $\equiv$ esPar?(n)

Extender el tipo NAT con la operación *esPar?* que indica si un número es o no es par.

esPar? : nat $\longrightarrow$ bool { }

esPar?(0) $\equiv$ true

esPar?(suc(n)) $\equiv$ $\neg$ esPar?(n)

Podemos ahora agregar *esImpar?* de la siguiente manera:

esImpar? : nat $\longrightarrow$ bool { }

esImpar?(0) $\equiv$ false

esImpar?(suc(n)) $\equiv$ esPar?(n)

- ¿Son recursivas estas definiciones?

Recursión

Extender el tipo Nat con la operación *esPar?* que indica si un número es o no es par.

esPar? : nat $\longrightarrow$ bool { }

esPar?(0) $\equiv$ true

esPar?(suc(n)) $\equiv$ esImpar?(n)

Podemos ahora agregar *esImpar?* de la siguiente manera:

esImpar? : nat $\longrightarrow$ bool { }

esImpar?(0) $\equiv$ false

esImpar?(suc(n)) $\equiv$ esPar?(n)

- ¿Son recursivas estas definiciones?
- ¿Está bien si modificamos "*esPar?*" de esta forma?

Recursión

Extender el tipo Nat con la operación *esPar?* que indica si un número es o no es par.

esPar? : nat $\longrightarrow$ bool { }

esPar?(0) $\equiv$ true

esPar?(suc(n)) $\equiv$ esImpar?(n)

Podemos ahora agregar *esImpar?* de la siguiente manera:

esImpar? : nat $\longrightarrow$ bool { }

esImpar?(0) $\equiv$ false

esImpar?(suc(n)) $\equiv$ esPar?(n)

- ¿Son recursivas estas definiciones?
- ¿Está bien si modificamos "*esPar?*" de esta forma?
- ¿Son recursivas estas definiciones?

Recursión INDIRECTA

- Esto se conoce como **recursión indirecta** (o **cíclica**).
- No tiene sentido para el caso de "*esPar?*" y "*esImpar?*" (!)
- En un rato vamos a ver algunos casos en los que sí tiene sentido...

Ejercicios (1)

Extender el tipo $\textsc{Secuencia}(\alpha)$ con la operación *primeros(n,s)* que devuelve los primeros *n* elementos de una secuencia *s*.

$\mathsf{primeros} \;:\; \mathsf{nat} \times \mathsf{secu}(\alpha) \longrightarrow \mathsf{secu}(\alpha)$ { ?? }

Ejercicios (1)

Extender el tipo Secuencia(α) con la operación *primeros(n,s)* que devuelve los primeros *n* elementos de una secuencia *s*.

primeros : nat n $\times$ secu(α) s $\longrightarrow$ secu(α) $\quad$ { n $\leq$ long(s) }

primeros(n, s) $\equiv$ **if** n = 0? **then**
<>
else
prim(s) • primeros(pred(n), fin(s))
fi

Ejercicios (2)

Extender el tipo Secuencia(α) con la operación *subcadenas(s,n)* que devuelve un conjunto con todas las subcadenas de *n* elementos contiguos de una secuencia *s*.

> *Por ejemplo, subcadenas*$([1, 2, 3, 4, 5],\ 3)$ *devuelve el conjunto* $\{[1, 2, 3], [2, 3, 4], [3, 4, 5]\}$.

subcadenas : secu(α) $\times$ nat $\longrightarrow$ conj(secu(α)) { ?? }

Ejercicios (2)

Extender el tipo Secuencia(α) con la operación *subcadenas(s,n)* que devuelve un conjunto con todas las subcadenas de *n* elementos contiguos de una secuencia *s*.

> *Por ejemplo, subcadenas*$([1, 2, 3, 4, 5],\ 3)$ *devuelve el conjunto* $\{[1, 2, 3], [2, 3, 4], [3, 4, 5]\}$.

subcadenas : secu(α) $\times$ nat $\longrightarrow$ conj(secu(α)) { }

```
subcadenas(s, n) ≡ if n > long(s) then
                       ∅
                   else
                       if n = 0? then
                           Ag(<>, ∅)
                       else
                           Ag(primeros(n,s), subcadenas(n, fin(s)))
                       fi
                   fi
```

Ejercicios (3)

Extender el tipo Secuencia(α) con *subsecuencias(s,n)* que devuelve un conjunto con todas las subsecuencias de *n* elementos (no necesariamente contiguos) de una secuencia *s*.

> *Por ejemplo, subsecuencias*$([1,2,3,4,5],\ 2)$ *devuelve el conjunto* $\{[1,2],[1,3],[1,4],[1,5],[2,3],[2,4],[2,5],[3,4],[3,5],[4,5]\}$.

subsecuencias : secu(α) $\times$ nat $\longrightarrow$ conj(secu(α)) { ?? }

Ejercicios (3)

subsecuencias : secu(α) $\times$ nat $\longrightarrow$ conj(secu(α)) { }

subsecuencias(s, n) $\equiv$ **if** n $>$ long(s) **then**
$\emptyset$
else
if n $=$ 0? **then**
Ag(<>, $\emptyset$)
else
agregarAd(prim(s), subsecuencias(fin(s), pred(n))) $\cup$
subsecuencias(fin(s), n)
fi
fi

agregarAd : α $\times$ conj(secu(α)) $\longrightarrow$ conj(secu(α)) { }

agregarAd(a, c) $\equiv$ **if** vacio?(c) **then**
$\emptyset$
else
Ag(a $\bullet$ dameUno(c), agregarAd(a, sinUno(c)))
fi

TAD RoseTree(α)

TAD Rosetree(α)

géneros rosetree(α)

observadores básicos

raíz : rosetree(α) $\longrightarrow$ α

hijos : rosetree(α) $\longrightarrow$ secu(rosetree(α))

generadores

rose : $\alpha \times$ secu(rosetree(α)) $\longrightarrow$ rosetree(α)

axiomas $\forall s$: secu(rosetree(α)) $\forall a$: α

raíz(rose(a, s)) $\equiv$ a

hijos(rose(a, s)) $\equiv$ s

Fin TAD

TAD RoseTree(α)

TAD Rosetree(α)

géneros rosetree(α)

observadores básicos

raíz : rosetree(α) $\longrightarrow$ α

hijos : rosetree(α) $\longrightarrow$ secu(rosetree(α))

generadores

rose : $\alpha \times$ secu(rosetree(α)) $\longrightarrow$ rosetree(α)

axiomas $\forall s$: secu(rosetree(α)) $\forall a$: α

raíz(rose(a, s)) $\equiv$ a

hijos(rose(a, s)) $\equiv$ s

Fin TAD

Ejercicio: Especificar la operación *incrementar(r,n)* que toma un *rosetree(nat) r* y un *nat n* y devuelve un árbol igual a *r* pero con todos los valores incrementados en *n*.

TAD RoseTree(α) - Ejercicios (1)

Especificar la operación *incrementar(r,n)* que toma un *rosetree(nat)* *r* y un *nat* *n* y devuelve un árbol igual a *r* pero con todos los valores incrementados en *n*.

incrementar : rosetree(nat) $\times$ nat $\longrightarrow$ rosetree(nat) { }

incrementar(r, n) $\equiv$ rose(raíz(r) + n, incrTodos(hijos(r), n))

incrTodos : secu(rosetree(nat)) $\times$ nat $\longrightarrow$ secu(rosetree(nat)) { }

incrTodos(sr, n) $\equiv$ **if** vacía?(sr) **then**
<>
else
incrementar(prim(sr),n) • incrTodos(fin(sr),n)
fi

TAD RoseTree(α) - Ejercicios (1)

Especificar la operación *incrementar(r,n)* que toma un *rosetree(nat)* *r* y un *nat n* y devuelve un árbol igual a *r* pero con todos los valores incrementados en *n*.

incrementar : rosetree(nat) $\times$ nat $\longrightarrow$ rosetree(nat) { }

incrementar(r, n) $\equiv$ rose(raíz(r) + n, incrTodos(hijos(r), n))

incrTodos : secu(rosetree(nat)) $\times$ nat $\longrightarrow$ secu(rosetree(nat)) { }

incrTodos(sr, n) $\equiv$ **if** vacía?(sr) **then**
<>
else
incrementar(prim(sr),n) • incrTodos(fin(sr),n)
fi

¿Son **recursivas** estas funciones?

TAD RoseTree(α) - Ejercicios (2)

Especificar la operación *sumaDeNiveles(r)* que toma un *rosetree(nat)* y devuelve una *secu(nat)* con las sumas de los valores en los nodos de cada nivel del rosetree (¡dibujito!).

sumaDeNiveles : rosetree(nat) $\longrightarrow$ secu(nat) { }

TAD RoseTree(α) - Ejercicios (2)

Especificar la operación *sumaDeNiveles(r)* que toma un *rosetree(nat)* y devuelve una *secu(nat)* con las sumas de los valores en los nodos de cada nivel del rosetree (¡dibujito!).

sumaDeNiveles : rosetree(nat) $\longrightarrow$ secu(nat) { }

sumaDeNiveles(r) $\equiv$ raíz(r) • sumarSecus(sumasNivelesTodos(hijos(r)))

sumasNivelesTodos : secu(rose(nat)) $\longrightarrow$ secu(secu(nat)) { }

sumasNivelesTodos(sr) $\equiv$ **if** vacía?(sr) **then**
<>
else
sumaDeNiveles(prim(sr)) • sumasNivelesTodos(fin(sr))
fi

sumarSecus : secu(secu(nat)) $\longrightarrow$ secu(nat) { }

sumarSecus(ss) $\equiv$ **if** vacía?(ss) **then**
<>
else
prim(ss) + sumarSecus(fin(ss))
fi

Observación: La suma de secuencias suma posición a posición (¡queda de tarea!).

Ejercicio: La fábrica de empanadas

Enunciado

Se quiere especificar el comportamiento de una fábrica de empanadas que está totalmente automatizada. A medida que se encuentran listas, las empanadas van saliendo de una máquina una a una y son depósitadas en una caja para empanadas. En la caja caben 12 empanadas y cuando ésta se llena, es automáticamente despachada y reemplazada por una caja vacía.

Ejercicio: La fábrica de empanadas

Enunciado

Se quiere especificar el comportamiento de una fábrica de empanadas que está totalmente automatizada. A medida que se encuentran listas, las empanadas van saliendo de una máquina una a una y son depósitadas en una caja para empanadas. En la caja caben 12 empanadas y cuando ésta se llena, es automáticamente despachada y reemplazada por una caja vacía.

Requerimientos

Se quiere saber:

- Cuántas cajas de empanadas se despacharon en total
- Cuántas empanadas hay en la caja que está actualmente abierta.

Resolución

Tenemos que definir:

Resolución

Tenemos que definir:

1. Los observadores y la igualdad observacional.
2. Los generadores.
3. Las otras operaciones (si hay).
4. Axiomatizar todo.

¿Qué queremos modelar?

1. ¿Qué nos están pidiendo sobre las empanadas? ¿Identificamos los gustos?

¿Qué queremos modelar?

1. ¿Qué nos están pidiendo sobre las empanadas? ¿Identificamos los gustos?
2. ¿Y para las cajas?

¿Qué queremos modelar?

1. ¿Qué nos están pidiendo sobre las empanadas? ¿Identificamos los gustos?
2. ¿Y para las cajas?
3. Alcanza con saber cuántas cajas salieron y cuántas empanadas hay en la caja actual...

¿Qué queremos modelar?

1. ¿Qué nos están pidiendo sobre las empanadas? ¿Identificamos los gustos?
2. ¿Y para las cajas?
3. Alcanza con saber cuántas cajas salieron y cuántas empanadas hay en la caja actual...

Entonces los observadores elegidos son:

cajasDespachadas	:	fabrica	$\longrightarrow$	nat
empanadasEnCaja	:	fabrica	$\longrightarrow$	nat

Generadores - Propuesta 1

crearFabrica	:	$\longrightarrow$	fabrica
caeEmpanada	: fabrica	$\longrightarrow$	fabrica
despacharCaja	: fabrica	$\longrightarrow$	fabrica

¿Qué problemas tiene esto?

Generadores - Propuesta 1

crearFabrica	:	$\longrightarrow$	fabrica	
caeEmpanada	: fabrica f	$\longrightarrow$	fabrica	$\{$ empanadasEnCaja(f) < 12 $\}$
despacharCaja	: fabrica f	$\longrightarrow$	fabrica	$\{$empanadasEnCaja(f) $= 12\}$

¿Con eso cierra?

Generadores - Propuesta 1

crearFabrica	:	$\longrightarrow$	fabrica	
caeEmpanada	: fabrica f	$\longrightarrow$	fabrica	$\{$ empanadasEnCaja(f) < 12 $\}$
despacharCaja	: fabrica f	$\longrightarrow$	fabrica	$\{$empanadasEnCaja(f) $= 12\}$

¿Con eso cierra?

- El enunciado dice que cuando la caja actual se llena "es automáticamente despachada y reemplazada por una caja vacía".
  - Estamos permitiendo que esto no ocurra...
  - Y estamos especificando que el despacho de la caja es una acción a realizar (si no, ¡no pueden caer más empanadas!).

Generadores - Propuesta 1

crearFabrica	:	$\longrightarrow$	fabrica	
caeEmpanada	: fabrica f	$\longrightarrow$	fabrica	$\{$ empanadasEnCaja(f) < 12 $\}$
despacharCaja	: fabrica f	$\longrightarrow$	fabrica	$\{$empanadasEnCaja(f) $= 12\}$

¿Con eso cierra?

- El enunciado dice que cuando la caja actual se llena "es automáticamente despachada y reemplazada por una caja vacía".
  - Estamos permitiendo que esto no ocurra...
  - Y estamos especificando que el despacho de la caja es una acción a realizar (si no, ¡no pueden caer más empanadas!).
- No estaríamos modelando el **comportamiento automático** en el TAD.

Generadores - Propuesta 1

crearFabrica	:	$\longrightarrow$	fabrica	
caeEmpanada	: fabrica f	$\longrightarrow$	fabrica	$\{$ empanadasEnCaja(f) < 12 $\}$
despacharCaja	: fabrica f	$\longrightarrow$	fabrica	$\{$empanadasEnCaja(f) $= 12\}$

¿Con eso cierra?

- El enunciado dice que cuando la caja actual se llena "es automáticamente despachada y reemplazada por una caja vacía".
  - Estamos permitiendo que esto no ocurra...
  - Y estamos especificando que el despacho de la caja es una acción a realizar (si no, ¡no pueden caer más empanadas!).
- No estaríamos modelando el **comportamiento automático** en el TAD.
- ¡No es necesario el generador *despacharCaja*!

Generadores

Los generadores son entonces:

crearFabrica	:	$\longrightarrow$	fabrica	
caeEmpanada	:	fabrica	$\longrightarrow$	fabrica

Axiomaticemos los observadores:

cajasDespachadas : fabrica $\longrightarrow$ nat $\{\ \}$

cajasDespachadas(crearFabrica) $\equiv$??

cajasDespachadas(caeEmpanada(f)) $\equiv$??

empanadasEnCaja : fabrica $\longrightarrow$ nat $\{\ \}$

empanadasEnCaja(crearFabrica) $\equiv$??

empanadasEnCaja(caeEmpanada(f)) $\equiv$??

Comportamiento automático (axiomas)

cajasDespachadas : fabrica $\longrightarrow$ nat $\{\ \}$

cajasDespachadas(crearFabrica) $\equiv$ 0

cajasDespachadas(caeEmpanada(f)) $\equiv$ **if** empanadasEnCaja(f) == 11 **then**
    cajasDespachadas(f) + 1
else
    cajasDespachadas(f)
fi

empanadasEnCaja : fabrica $\longrightarrow$ nat $\{\ \}$

empanadasEnCaja(crearFabrica) $\equiv$ 0

empanadasEnCaja(caeEmpanada(f)) $\equiv$ **if** empanadasEnCaja(f) == 11 **then**
    0
else
    empanadasEnCaja(f) + 1
fi

Reflexiones

1. ¿Qué impacto tuvo el comportamiento automático en la elección de los *observadores*?

Reflexiones

1. ¿Qué impacto tuvo el comportamiento automático en la elección de los *observadores*?

   Ninguno: Para distinguir lo que diferenciaba a las fábricas no importó si el comportamiento era automático o no.

Reflexiones

1. ¿Qué impacto tuvo el comportamiento automático en la elección de los *observadores*?

   Ninguno: Para distinguir lo que diferenciaba a las fábricas no importó si el comportamiento era automático o no.

2. ¿Qué impacto tuvo el comportamiento automático en los *generadores*?

Reflexiones

1. ¿Qué impacto tuvo el comportamiento automático en la elección de los *observadores*?

   Ninguno: Para distinguir lo que diferenciaba a las fábricas no importó si el comportamiento era automático o no.

2. ¿Qué impacto tuvo el comportamiento automático en los *generadores*?

   Algo: Tuvimos que quitar el generador que despachaba la caja (ya que eso era parte del comportamiento automático).

Reflexiones

1. ¿Qué impacto tuvo el comportamiento automático en la elección de los *observadores*?

   Ninguno: Para distinguir lo que diferenciaba a las fábricas no importó si el comportamiento era automático o no.

2. ¿Qué impacto tuvo el comportamiento automático en los *generadores*?

   Algo: Tuvimos que quitar el generador que despachaba la caja (ya que eso era parte del comportamiento automático).

3. ¿Qué impacto tuvo el comportamiento automático en los *axiomas*?

Reflexiones

1. ¿Qué impacto tuvo el comportamiento automático en la elección de los *observadores*?

   Ninguno: Para distinguir lo que diferenciaba a las fábricas no importó si el comportamiento era automático o no.

2. ¿Qué impacto tuvo el comportamiento automático en los *generadores*?

   Algo: Tuvimos que quitar el generador que despachaba la caja (ya que eso era parte del comportamiento automático).

3. ¿Qué impacto tuvo el comportamiento automático en los *axiomas*?

   Mucho: El comportamiento queda plenamente descripto en los axiomas.

Resumen

Cuando alguna parte del comportamiento del TAD debe ser automática:

- No deberíamos especificar una acción "manual" para este comportamiento (a menos que también sea el caso).
- No deberíamos permitir que existan instancias del TAD en las que el comportamiento debería haberse aplicado y no lo hizo.
- No deberíamos restringir acciones que requieran que suceda el comportamiento, ya que éste es automático (por ejemplo, la precondición de *caeEmpanada*).

Ejercicio: El ascensor inútil

Enunciado

Se quiere especificar el comportamiento de un ascensor que lleva personas entre dos pisos. La capacidad máxima del ascensor es de 3 personas. El ascensor se pone en funcionamiento cuando ingresan 3 personas, sin necesidad de apretar ningún botón. Cuando el ascensor se pone en movimiento, se desplaza del piso en el que está hacia el otro. En cuanto llega al piso de destino, las personas que están en el interior del ascensor lo desocupan inmediatamente. En cualquier momento pueden llegar personas a cualquiera de los dos pisos. En el caso que el ascensor no esté, las personas forman una fila y esperan a que el ascensor las venga a buscar.

Ejercicio: El ascensor inútil

Enunciado

Se quiere especificar el comportamiento de un ascensor que lleva personas entre dos pisos. La capacidad máxima del ascensor es de 3 personas. El ascensor se pone en funcionamiento cuando ingresan 3 personas, sin necesidad de apretar ningún botón. Cuando el ascensor se pone en movimiento, se desplaza del piso en el que está hacia el otro. En cuanto llega al piso de destino, las personas que están en el interior del ascensor lo desocupan inmediatamente. En cualquier momento pueden llegar personas a cualquiera de los dos pisos. En el caso que el ascensor no esté, las personas forman una fila y esperan a que el ascensor las venga a buscar.

Requerimientos

Se quiere saber:

- En qué piso está el ascensor.
- La cantidad de personas esperando en cada piso.

TAD Ascensor - Resolución

Igual que antes, tenemos que definir:

1. Los observadores y la igualdad observacional.
2. Los generadores.
3. Las otras operaciones (si hay).
4. Axiomatizar todo.

TAD Ascensor - ¿Qué queremos modelar?

1. ¿Qué nos están pidiendo sobre las personas? ¿Nos interesa identificarlas?
2. ¿Hace falta modelar la fila de personas?
3. ¿Nos interesa diferenciar a las personas que están esperando dentro y fuera del ascensor?

TAD Ascensor - ¿Qué queremos modelar?

1. ¿Qué nos están pidiendo sobre las personas? ¿Nos interesa identificarlas?
2. ¿Hace falta modelar la fila de personas?
3. ¿Nos interesa diferenciar a las personas que están esperando dentro y fuera del ascensor?

Entonces los observadores elegidos son:

$$\begin{array}{lll} \text{personasEn} & : & \text{ascensor} \times \text{piso} \longrightarrow \text{nat} \\ \text{piso?} & : & \text{ascensor} \longrightarrow \text{piso} \end{array}$$

Supongamos que el tipo *piso* tiene dos valores posibles A y B y la operación:

$$\begin{array}{lll} \text{otroPiso} & : & \text{piso} \longrightarrow \text{piso} \\ \text{otroPiso(A)} & \equiv & \text{B} \\ \text{otroPiso(B)} & \equiv & \text{A} \end{array} \qquad \{\ \}$$

TAD Ascensor - Generadores

Pensemos el conjunto de generadores del TAD Ascensor...

TAD Ascensor - Generadores

Pensemos el conjunto de generadores del TAD Ascensor...

crearAscensor	:	$\longrightarrow$	ascensor			
llegaPersona	:	ascensor $\times$ piso		$\longrightarrow$	ascensor	

TAD Ascensor - Generadores

Pensemos el conjunto de generadores del TAD Ascensor...

crearAscensor	:	$\longrightarrow$	ascensor	
llegaPersona	:	ascensor $\times$ piso	$\longrightarrow$	ascensor

Axiomaticemos los observadores:

personasEn : ascensor $\times$ piso $\longrightarrow$ nat $\{\ \}$

personasEn(crearAscensor, p)) $\equiv$??

personasEn(llegaPersona(a,p'), p) $\equiv$??

piso? : ascensor $\longrightarrow$ piso $\{\ \}$

piso?(crearAscensor) $\equiv$??

piso?(llegaPersona(a,p)) $\equiv$??

TAD Ascensor - Axiomas (1)

personasEn : ascensor × piso $\longrightarrow$ nat $\{\ \}$

personasEn(crearAscensor, p)) $\equiv$ 0

personasEn(llegaPersona(a,p'), p) $\equiv$ **if** p' = piso?(a) $\wedge$ personasEn(a,p') = 2 **then**
if p = p' **then**
0
else
if peronasEn(a,p) < 3 **then**
peronasEn(a,p)
else
peronasEn(a,p) - 3
fi
fi
else
if p = p' **then**
personasEn(a, p) + 1
else
personasEn(a, p)
fi
fi

TAD Ascensor - Axiomas (2)

piso? : ascensor $\longrightarrow$ piso $\{\ \}$

piso?(crearAscensor) $\equiv$ A

piso?(llegaPersona(a,p)) $\equiv$ **if** p $=$ otroPiso(piso?(a)) $\vee$ personasEn(a,piso?(a)) $<$ 2
then
    piso?(a)
else
    if personasEn(a,otroPiso(piso?(a))) $<$ 3 **then**
        otroPiso(piso?(a))
    else
        piso?(a)
    fi
fi

Observación: en *piso?* y *personasEn*, tenemos recursión directa e indirecta.

Resumen (¡otra vez!)

Cuando alguna parte del comportamiento del TAD debe ser automática:

- No deberíamos especificar una acción "manual" para este comportamiento (a menos que también sea el caso).
- No deberíamos permitir que existan instancias del TAD en las que el comportamiento debería haberse aplicado y no lo hizo.
- No deberíamos restringir acciones que requieran que suceda el comportamiento, ya que éste es automático (por ejemplo, la precondición de *caeEmpanada*).

Ejercicio (de tarea): El ascensor 2.0

Enunciado

Muchas veces la gente se cansa de esperar el ascensor, dado que mientras haya menos de 3 personas en el piso donde está el ascensor, éste no va a moverse. Esto implica que en el otro piso pueden acumularse muchísimas personas esperando.
Para esto se decidió incorporar un botón en cada piso para llamar al ascensor. Si el botón es presionado en el piso donde el ascensor se encuentra, nada sucede. Por el contrario, si se presiona en el piso donde el ascensor no está, el ascensor se mueve hacia el piso destino llevando con él a todas las personas que su capacidad le permita y que hayan estado esperando en el piso de origen.
Obviamente, si el botón es presionado es que hay al menos una persona en el piso donde se lo presionó.