Introducción a Tipos Abstractos de Datos

Algoritmos y Estructuras de Datos II

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

23 de agosto de 2019

Objetivo: aprender a escribir especificaciones como esta

TAD NAT

géneros nat

exporta nat, generadores, observadores, $+$, $-$, $\times$, $<$, $\leq$, mín, máx

usa BOOL

igualdad observacional

$$(\forall n, m : \text{nat}) \left(n =_{\text{obs}} m \iff \left(\begin{array}{l} (n = 0? =_{\text{obs}} m = 0?) \wedge_{\text{L}} \\ (\neg(n = 0?) \Rightarrow_{\text{L}} (\text{pred}(n) =_{\text{obs}} \text{pred}(m))) \end{array} \right) \right)$$

observadores básicos

$\bullet = 0?$	: nat	$\longrightarrow$	bool	
pred	: nat n	$\longrightarrow$	nat	$\{\neg(n = 0?)\}$

generadores

0	:	$\longrightarrow$	nat	
suc	: nat	$\longrightarrow$	nat	

otras operaciones

$\bullet + \bullet$	: nat $\times$ nat	$\longrightarrow$	nat	
$\bullet - \bullet$	: nat $n \times$ nat m	$\longrightarrow$	nat	$\{m \leq n\}$
$\bullet \times \bullet$	: nat $\times$ nat	$\longrightarrow$	nat	

axiomas $\forall\, n, m$: nat

$0 = 0?$	$\equiv$	true
$\text{suc}(n) = 0?$	$\equiv$	false
$\text{pred}(\text{suc}(n))$	$\equiv$	n
$n + m$	$\equiv$	**if** $m = 0?$ **then** n **else** $\text{suc}(n + \text{pred}(m))$ **fi**

Pero primero necesitamos hacer un repaso de recursión.

¿Qué es la recursión?

- ¿Qué es una función recursiva?

¿Qué es la recursión?

- ¿Qué es una función recursiva?

  Una función que en su definición se llama a sí misma.

¿Qué es la recursión?

- ¿Qué es una función recursiva?

  Una función que en su definición se llama a sí misma.

- En general, una función recursiva tiene,
  - uno o más *casos base*
  - uno o más *casos recursivos*

¿Qué es la recursión?

- ¿Qué es una función recursiva?

  Una función que en su definición se llama a sí misma.

- En general, una función recursiva tiene,
  - uno o más *casos base*
  - uno o más *casos recursivos*

- ¿Por qué tiene sentido hacer recursión?
  (¿Por qué no se cuelga el programa?)

¿Qué es la recursión?

- ¿Qué es una función recursiva?

  Una función que en su definición se llama a sí misma.

- En general, una función recursiva tiene,
  - uno o más *casos base*
  - uno o más *casos recursivos*

- ¿Por qué tiene sentido hacer recursión?
  (¿Por qué no se cuelga el programa?)

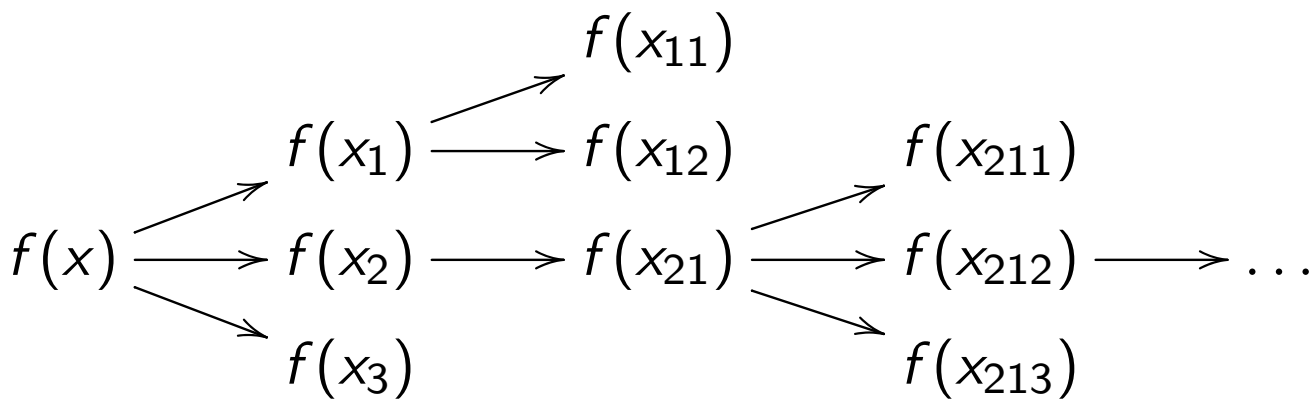
  Tiene al menos un *caso base*.

  Cada llamado recursivo "nos acerca" un poco al caso base.

¿Cómo llevarse bien con la recursión?

Queremos que cada llamado recursivo se haga sobre un caso "más chico", que eventualmente llegue a un elemento mínimo que corresponde a un *caso base*.

Gráficamente



¿Cómo llevarse bien con la recursión?

¿Cómo podemos asegurarnos de que una función recursiva resuelve el problema deseado?

- Tenemos que asegurarnos de algunas cosas:

¿Cómo llevarse bien con la recursión?

¿Cómo podemos asegurarnos de que una función recursiva resuelve el problema deseado?

- Tenemos que asegurarnos de algunas cosas:
  1. La función recursiva tiene que terminar. Los llamados recursivos deben *achicar el tamaño* del problema a resolver, de acuerdo con alguna relación de orden bien fundado.

     ¿Qué falta?

¿Cómo llevarse bien con la recursión?

¿Cómo podemos asegurarnos de que una función recursiva resuelve el problema deseado?

- Tenemos que asegurarnos de algunas cosas:
  1. La función recursiva tiene que terminar. Los llamados recursivos deben *achicar el tamaño* del problema a resolver, de acuerdo con alguna relación de orden bien fundado.

     ¿Qué falta?

  2. Los casos base deben resolver el problema.

¿Cómo llevarse bien con la recursión?

¿Cómo podemos asegurarnos de que una función recursiva resuelve el problema deseado?

- Tenemos que asegurarnos de algunas cosas:
  1. La función recursiva tiene que terminar. Los llamados recursivos deben *achicar el tamaño* del problema a resolver, de acuerdo con alguna relación de orden bien fundado.

     ¿Qué falta?

  2. Los casos base deben resolver el problema.
  3. Los casos recursivos deben resolver el problema, suponiendo que cada llamado recursivo resuelve el correspondiente subproblema.

- Si se cumplen esas tres condiciones, la función queda bien definida y resuelve el problema deseado.
  ¿Por qué? ¿Cómo nos convencemos de esto?
  ¿Qué se aplica para demostrar que es correcto?

¿Dónde vamos a usar recursión?

- Vamos a usar recursión para axiomatizar el comportamiento de las funciones de los TADs.
- Antes de empezar a practicar, repasemos los TADs básicos del apunte.

Ejercicio: Reverso

Extender el tipo $\textsc{Secuencia}(\alpha)$ con la operación *reverso* que devuelve la misma secuencia en orden inverso.

Ejercicio: Reverso

Extender el tipo $\textsc{Secuencia}(\alpha)$ con la operación *reverso* que devuelve la misma secuencia en orden inverso.

$\mathsf{reverso} \ : \ \mathsf{secu}(\alpha) \ \longrightarrow \ \mathsf{secu}(\alpha)$ $\{\ \}$

$\mathsf{reverso}(<>) \ \equiv \ <>$

$\mathsf{reverso}(a \bullet s) \ \equiv \ \mathsf{reverso}(s) \circ a$

Ejercicio: esPrefijo?

Extender el tipo $\textsc{Secuencia}(\alpha)$ con la operación *esPrefijo?(s, t)* que verifica si la secuencia s es prefijo de la secuencia t.

Ejercicio: esPrefijo?

Extender el tipo $\textsc{Secuencia}(\alpha)$ con la operación *esPrefijo?(s, t)* que verifica si la secuencia s es prefijo de la secuencia t.

esPrefijo? : secu(α) $\times$ secu(α) $\longrightarrow$ bool $\quad \{\ \}$

esPrefijo?($<>, t$) $\equiv$ *true*

esPrefijo?($a \bullet s, t$) $\equiv$ $\neg$ vacía?(t) $\wedge_{\mathrm{L}}$ prim(t) $= a \wedge$ esPrefijo?(s,fin(t))

Ejercicio: Reemplazar

Extender el tipo $\textsc{Secuencia}(\alpha)$ con la operación *reemplazar(s, a, b)* que reemplaza en la secuencia *s* todas las apariciones del elemento *a* por el elemento *b*.

Ejercicio: Reemplazar

Extender el tipo $\textsc{Secuencia}(\alpha)$ con la operación *reemplazar(s, a, b)* que reemplaza en la secuencia *s* todas las apariciones del elemento *a* por el elemento *b*.

$\text{reemplazar} : \text{secu}(\alpha) \times \alpha \times \alpha \longrightarrow \text{secu}(\alpha)$ $\{\ \}$

$\text{reemplazar}(<>, a, b) \equiv <>$

$\text{reemplazar}(t \bullet s, a, b) \equiv (\textbf{if } \text{t} = a \textbf{ then } b \textbf{ else } \text{t} \textbf{ fi}) \bullet \text{reemplazar}(s, a, b)$

Ejercicio: #Apariciones

Definir la operación *#Apariciones(ab, a)* sobre el TAD $\text{AB}(\alpha)$ (árboles binarios) que devuelve la cantidad de apariciones del elemento *a* en el árbol *ab*.

Ejercicio: #Apariciones

Definir la operación *#Apariciones(ab, a)* sobre el TAD $\text{AB}(\alpha)$ (árboles binarios) que devuelve la cantidad de apariciones del elemento *a* en el árbol *ab*.

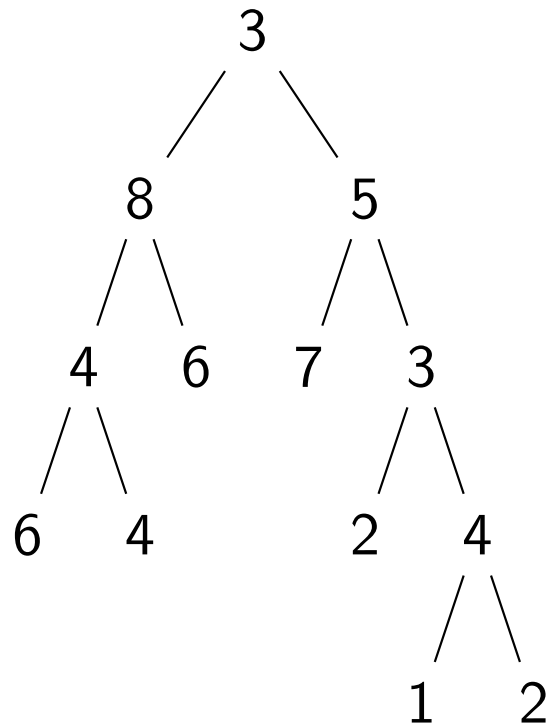
$$\#\text{Apariciones} : \text{ab}(\alpha) \times \alpha \longrightarrow \text{nat} \qquad \{\ \}$$

$$\#\text{Apariciones}(\mathit{nil}, a) \equiv 0$$

$$\#\text{Apariciones}(\mathit{bin}(i, r, d), a) \equiv \textbf{if } r = a \textbf{ then } 1 \textbf{ else } 0 \textbf{ fi} + \#\text{Apariciones}(i, a) + \#\text{Apariciones}(d, a)$$

Ejercicio: últimoNivelCompleto

Definir la operación *últimoNivelCompleto* sobre el TAD $\text{AB}(\alpha)$ (árboles binarios) que devuelve el número del último nivel que está completo (es decir, aquél que tiene todos los nodos posibles).



Ejercicio: últimoNivelCompleto

Definir la operación *últimoNivelCompleto* sobre el TAD $\text{AB}(\alpha)$ (árboles binarios) que devuelve el número del último nivel que está completo (es decir, aquél que tiene todos los nodos posibles).

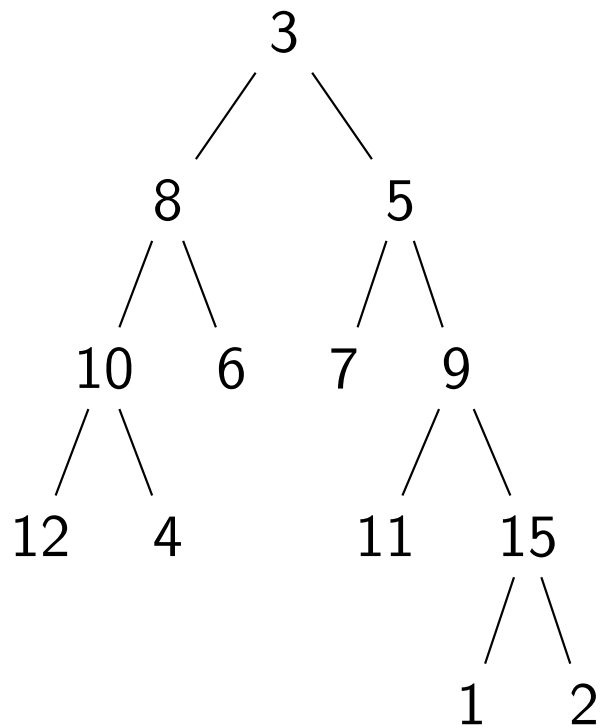
$$\text{últimoNivelCompleto} : \text{ab}(\alpha) \longrightarrow \text{nat} \qquad \{\ \}$$

$$\text{últimoNivelCompleto}(\mathit{nil}) \equiv 0$$

$$\text{últimoNivelCompleto}(\mathit{bin}(i, r, d)) \equiv \min(\text{últimoNivelCompleto}(i), \text{últimoNivelCompleto}(d)) + 1$$

Ejercicio: CaminoHasta

Definir la operación *CaminoHasta(ab, a)* sobre el TAD $\text{AB}(\alpha)$ (árboles binarios) que devuelve una secuencia que representa el camino desde la raíz del árbol *ab* hasta el elemento *a* en el mismo (asumir que el árbol no tiene elementos repetidos). Si el elemento *a* no aparece en el árbol, debe devolverse la secuencia vacía.



Ejercicio: CaminoHasta

Definir la operación *CaminoHasta(ab, a)* sobre el TAD $\text{AB}(\alpha)$ (árboles binarios) que devuelve una secuencia que representa el camino desde la raíz del árbol *ab* hasta el elemento *a* en el mismo (asumir que el árbol no tiene elementos repetidos). Si el elemento *a* no aparece en el árbol, debe devolverse la secuencia vacía.

CaminoHasta : ab(α) $\times$ α $\longrightarrow$ secu(α) { }

CaminoHasta(*nil*, *a*) $\equiv$ $<>$

CaminoHasta(*bin*(i, r, d), *a*) $\equiv$ **if** $r = a$ **then**
    $a \bullet <>$
else
    if #apariciones(i, *a*) > 0 **then**
        $r \bullet$ CaminoHasta(i, *a*)
    else
        if #apariciones(d, *a*) > 0 **then**
            $r \bullet$ CaminoHasta(d, *a*)
        else
            $<>$
        fi
    fi
fi

Tipos Abstractos de Datos

- ¿Qué es un tipo de datos?

Tipos Abstractos de Datos

- ¿Qué es un tipo de datos?

  Conjunto de valores dotado de operaciones.

Tipos Abstractos de Datos

- ¿Qué es un tipo de datos?

  Conjunto de valores dotado de operaciones.

- ¿Qué es un tipo abstracto de datos?

Tipos Abstractos de Datos

- ¿Qué es un tipo de datos?

  Conjunto de valores dotado de operaciones.

- ¿Qué es un tipo abstracto de datos?

  Es un tipo de datos especificado por su comportamiento.

Tipos Abstractos de Datos

- ¿Qué es un tipo de datos?

  Conjunto de valores dotado de operaciones.

- ¿Qué es un tipo abstracto de datos?

  Es un tipo de datos especificado por su comportamiento.

- Es decir, en un TADs se definen funcionalidades y se explica **qué** hace cada una, sin especificar **cómo** lo hace. (El "cómo" lo dejamos para más adelante).

Tipos Abstractos de Datos

- ¿Qué es un tipo de datos?

  Conjunto de valores dotado de operaciones.

- ¿Qué es un tipo abstracto de datos?

  Es un tipo de datos especificado por su comportamiento.

- Es decir, en un TADs se definen funcionalidades y se explica **qué** hace cada una, sin especificar **cómo** lo hace. (El "cómo" lo dejamos para más adelante).

- Esto facilita la resolución de problemas "grandes" modularizando en problemas de menor complejidad.

Bibliografía sobre TADs

Object-Oriented Software Construction.
Bertrand Meyer, Prentice-Hall, 1988.

- Capítulo 6 — *Abstract Data Types*.
  Para la parte de especificación.

- Capítulo 11 — *Design by Contract: building reliable software*.
  Para la parte de diseño.

Ejercicio: Agenda de compromisos

Necesitamos una agenda en la cual podamos registrar compromisos para un día. Por ejemplo, "Turno con el dentista de 16 a 17 hs", o "Reunión de cátedra de 18 a 20". No hay problema con que los compromisos registrados en la agenda se solapen. De hecho, nos interesa saber, dada una hora del día, qué compromisos tenemos en ese momento. Además, dado un intervalo de horas, quiséramos poder saber qué hora del intervalo es la más ocupada en la agenda, en caso de que haya más de una que cumpla esta condición queremos la que sea más temprano.

Resolución — en teoría

¿Cómo empezamos?

1. Leer bien el enunciado e identificar qué cosas son importantes y qué cosas no lo son.
2. Definir los observadores y la igualdad observacional.
3. Definir los generadores.
4. Definir las otras operaciones.
5. Definir las restricciones donde corresponda
6. Incluir otras operaciones auxiliares, de haberlas.
7. Axiomatizar todo.

Resolución — en teoría

¿Cómo empezamos?

1. Leer bien el enunciado e identificar qué cosas son importantes y qué cosas no lo son.
2. Definir los observadores y la igualdad observacional.
3. Definir los generadores.
4. Definir las otras operaciones.
5. Definir las restricciones donde corresponda
6. Incluir otras operaciones auxiliares, de haberlas.
7. Axiomatizar todo.

Pero, ¿se puede hacer así, realmente, paso por paso?

En general conviene ir pensando algunas cosas en simultáneo.

Repasemos el enunciado

Necesitamos una agenda en la cual podamos registrar compromisos para un día. Por ejemplo, "Turno con el dentista de 16 a 17 hs", o "Reunión de cátedra de 18 a 20hs". No hay problema con que los compromisos registrados en la agenda se solapen. De hecho, nos interesa saber, dada una hora del día, qué compromisos tenemos en ese momento. Además, dado un intervalo de horas, quiséramos poder saber qué hora del intervalo es la más ocupadas en la agenda, en caso de que haya más de una que cumpla esta condición queremos la que sea más temprano.

Repasemos el enunciado

Necesitamos una agenda en la cual podamos registrar compromisos para un día. Por ejemplo, "Turno con el dentista de 16 a 17 hs", o "Reunión de cátedra de 18 a 20hs". No hay problema con que los compromisos registrados en la agenda se solapen. De hecho, nos interesa saber, dada una hora del día, qué compromisos tenemos en ese momento. Además, dado un intervalo de horas, quiséramos poder saber qué hora del intervalo es la más ocupadas en la agenda, en caso de que haya más de una que cumpla esta condición queremos la que sea más temprano.

Repasemos el enunciado

Necesitamos una agenda en la cual podamos registrar compromisos para un día. Por ejemplo, "Turno con el dentista de 16 a 17 hs", o "Reunión de cátedra de 18 a 20hs". No hay problema con que los compromisos registrados en la agenda se solapen. De hecho, nos interesa saber, dada una hora del día, qué compromisos tenemos en ese momento. Además, dado un intervalo de horas, quiséramos poder saber qué hora del intervalo es la más ocupadas en la agenda, en caso de que haya más de una que cumpla esta condición queremos la que sea más temprano.

Repasemos el enunciado

Necesitamos una agenda en la cual podamos registrar compromisos para un día. Por ejemplo, "Turno con el dentista de 16 a 17 hs", o "Reunión de cátedra de 18 a 20hs". No hay problema con que los compromisos registrados en la agenda se solapen. De hecho, nos interesa saber, dada una hora del día, qué compromisos tenemos en ese momento. Además, dado un intervalo de horas, quiséramos poder saber qué hora del intervalo es la más ocupadas en la agenda, en caso de que haya más de una que cumpla esta condición queremos la que sea más temprano.

Repasemos el enunciado

Necesitamos una agenda en la cual podamos registrar compromisos para un día. Por ejemplo, "Turno con el dentista de 16 a 17 hs", o "Reunión de cátedra de 18 a 20hs". No hay problema con que los compromisos registrados en la agenda se solapen. De hecho, nos interesa saber, dada una hora del día, qué compromisos tenemos en ese momento. Además, dado un intervalo de horas, quiséramos poder saber qué hora del intervalo es la más ocupadas en la agenda, en caso de que haya más de una que cumpla esta condición queremos la que sea más temprano.

Repasemos el enunciado

Necesitamos una agenda en la cual podamos registrar compromisos para un día. Por ejemplo, "Turno con el dentista de 16 a 17 hs", o "Reunión de cátedra de 18 a 20hs". No hay problema con que los compromisos registrados en la agenda se solapen. De hecho, nos interesa saber, dada una hora del día, qué compromisos tenemos en ese momento. Además, dado un intervalo de horas, quiséramos poder saber qué hora del intervalo es la más ocupadas en la agenda, en caso de que haya más de una que cumpla esta condición queremos la que sea más temprano.

- La agenda debe permitir registrar compromisos (String), con su hora de inicio y su hora de fin (Nats).

Repasemos el enunciado

Necesitamos una agenda en la cual podamos registrar compromisos para un día. Por ejemplo, "Turno con el dentista de 16 a 17 hs", o "Reunión de cátedra de 18 a 20hs". No hay problema con que los compromisos registrados en la agenda se solapen. De hecho, nos interesa saber, dada una hora del día, qué compromisos tenemos en ese momento. Además, dado un intervalo de horas, quiséramos poder saber qué hora del intervalo es la más ocupadas en la agenda, en caso de que haya más de una que cumpla esta condición queremos la que sea más temprano.

- La agenda debe permitir registrar compromisos (String), con su hora de inicio y su hora de fin (Nats).
- Deberíamos poder consultar los compromisos de un determinado momento.

Repasemos el enunciado

Necesitamos una agenda en la cual podamos registrar compromisos para un día. Por ejemplo, "Turno con el dentista de 16 a 17 hs", o "Reunión de cátedra de 18 a 20hs". No hay problema con que los compromisos registrados en la agenda se solapen. De hecho, nos interesa saber, dada una hora del día, qué compromisos tenemos en ese momento. Además, dado un intervalo de horas, quiséramos poder saber qué hora del intervalo es la más ocupadas en la agenda, en caso de que haya más de una que cumpla esta condición queremos la que sea más temprano.

- La agenda debe permitir registrar compromisos (String), con su hora de inicio y su hora de fin (Nats).
- Deberíamos poder consultar los compromisos de un determinado momento.
- Saber qué horas de un intervalo son las más ocupadas.

Observadores, generadores e igualdad

- ¿Observadores?

Observadores, generadores e igualdad

- ¿Observadores?

compromisos : agenda $\times$ nat h $\longrightarrow$ conj(compromiso) $\quad \{0 \leq h < 24\}$

Observadores, generadores e igualdad

- ¿Observadores?

compromisos : agenda $\times$ nat h $\longrightarrow$ conj(compromiso) $\quad \{0 \leq h < 24\}$

- ¿Igualdad observacional?

Observadores, generadores e igualdad

- ¿Observadores?

$$\text{compromisos} : \text{agenda} \times \text{nat } h \longrightarrow \text{conj(compromiso)} \qquad \{0 \leq h < 24\}$$

- ¿Igualdad observacional?

$$(\forall a, a' : \text{agenda})$$
$$(\ a =_{\text{obs}} a' \Leftrightarrow (\forall h : \text{nat})\ (\text{compromisos}(a, h) =_{\text{obs}} \text{compromisos}(a', h)))$$

Observadores, generadores e igualdad

- ¿Observadores?

$$\text{compromisos} : \text{agenda} \times \text{nat } h \longrightarrow \text{conj(compromiso)} \qquad \{0 \leq h < 24\}$$

- ¿Igualdad observacional?

$$(\forall a, a' : \text{agenda})$$
$$(\ a =_{\text{obs}} a' \Leftrightarrow (\forall h : \text{nat})\ (\text{compromisos}(a, h) =_{\text{obs}} \text{compromisos}(a', h)))$$

- ¿Generadores?

Observadores, generadores e igualdad

- ¿Observadores?

compromisos : agenda $\times$ nat h $\longrightarrow$ conj(compromiso) $\quad \{0 \leq h < 24\}$

- ¿Igualdad observacional?

$(\forall a, a'$: agenda)
($a =_{\text{obs}} a' \Leftrightarrow (\forall h$: nat) (compromisos$(a, h) =_{\text{obs}}$ compromisos$(a', h)))$

- ¿Generadores?

crearAgenda : $\longrightarrow$ agenda
registrar : agenda $\times$ compromiso $\times$ nat $\times$ nat $\longrightarrow$ agenda

Observadores, generadores e igualdad

- ¿Observadores?

compromisos : agenda $\times$ nat h $\longrightarrow$ conj(compromiso) $\quad \{0 \leq h < 24\}$

- ¿Igualdad observacional?

$(\forall a, a'$: agenda)
($a =_{\text{obs}} a' \Leftrightarrow (\forall h$: nat) (compromisos$(a, h) =_{\text{obs}}$ compromisos$(a', h)))$

- ¿Generadores?

crearAgenda : $\longrightarrow$ agenda
registrar : agenda $\times$ compromiso $\times$ nat d $\times$ nat h $\longrightarrow$ agenda $\quad \{0 \leq d < h < 24\}$

Observadores, generadores e igualdad

- ¿Observadores?

compromisos : agenda $\times$ nat h $\longrightarrow$ conj(compromiso) $\quad \{0 \leq h < 24\}$

- ¿Igualdad observacional?

$(\forall a, a'$: agenda)
($a =_{\text{obs}} a' \Leftrightarrow (\forall h$: nat) (compromisos$(a, h) =_{\text{obs}}$ compromisos$(a', h)))$

- ¿Generadores?

crearAgenda : $\longrightarrow$ agenda
registrar : agenda $\times$ compromiso $\times$ nat d $\times$ nat h $\longrightarrow$ agenda $\quad \{0 \leq d < h < 24\}$

- ¿Otras operaciones?

Observadores, generadores e igualdad

- ¿Observadores?

compromisos : agenda $\times$ nat h $\longrightarrow$ conj(compromiso) $\quad \{0 \leq h < 24\}$

- ¿Igualdad observacional?

$(\forall a, a'$: agenda)
($a =_{\text{obs}} a' \Leftrightarrow (\forall h$: nat) (compromisos$(a, h) =_{\text{obs}}$ compromisos$(a', h)))$

- ¿Generadores?

crearAgenda : $\longrightarrow$ agenda
registrar : agenda $\times$ compromiso $\times$ nat d $\times$ nat h $\longrightarrow$ agenda $\quad \{0 \leq d < h < 24\}$

- ¿Otras operaciones?

primeraHoraMasOcupada : agenda $\times$ nat $\times$ nat $\longrightarrow$ nat

Observadores, generadores e igualdad

- ¿Observadores?

$$\text{compromisos} : \text{agenda} \times \text{nat } h \longrightarrow \text{conj(compromiso)} \quad \{0 \leq h < 24\}$$

- ¿Igualdad observacional?

$$(\forall a, a' : \text{agenda})$$
$$(\ a =_{\text{obs}} a' \Leftrightarrow (\forall h : \text{nat})\ (\text{compromisos}(a, h) =_{\text{obs}} \text{compromisos}(a', h)))$$

- ¿Generadores?

$$\text{crearAgenda} : \longrightarrow \text{agenda}$$
$$\text{registrar} : \text{agenda} \times \text{compromiso} \times \text{nat } d \times \text{nat } h \longrightarrow \text{agenda} \quad \{0 \leq d < h < 24\}$$

- ¿Otras operaciones?

$$\text{primeraHoraMasOcupada} : \text{agenda} \times \text{nat } d \times \text{nat } h \longrightarrow \text{nat} \quad \{0 \leq d \leq h < 24\}$$

Axiomas

compromisos : agenda $\times$ nat h $\longrightarrow$ conj(compromiso) $\{0 \leq h < 24\}$

compromisos(crearAgenda, h) $\equiv$???

compromisos(registrar(a, ini, fin, c), h) $\equiv$???

primeraHoraMasOcupada : agenda $\times$ nat d $\times$ nat h $\longrightarrow$ nat $\{0 \leq d < h < 24\}$

primHoraMasOcupada(a, d, h) $\equiv$???

Axiomas

compromisos : agenda $\times$ nat h $\longrightarrow$ conj(compromiso) $\{0 \leq h < 24\}$

compromisos(crearAgenda, h) $\equiv$ $\emptyset$

compromisos(registrar(a, ini, fin, c), h) $\equiv$ **if** $ini \leq h < fin$ **then**
Ag(c, compromisos(a, h))
else
compromisos(a, h)
fi

primeraHoraMasOcupada : agenda $\times$ nat d $\times$ nat h $\longrightarrow$ nat $\{0 \leq d < h < 24\}$

primHoraMasOcupada(a, d, h) $\equiv$ **if** $d = h$ **then**
d
else
if #compromisos(a, primHoraMasOcupada(a, $d + 1$, h)) $>$ #compromisos(a, d) **then**
primHoraMasOcupada(a, $d + 1$, h)
else
d
fi
fi

Ejercicio: Insoportables

Insoportables es un programa televisivo muy exitoso que sale al aire todas las noches; en él se debate acerca de las relaciones entre los personajes de la farándula (los "famosos"). Con el tiempo, distintos famosos se van incorporando al programa (y nunca dejan de pertenecer al mismo).

Debido a la gran cantidad de peleas y reconciliaciones, los productores nos encargaron el desarrollo de un sistema que permita saber en todo momento quiénes están peleados y quiénes no. Además, los productores quieren poder determinar quién es el famoso que actualmente está involucrado en la mayor cantidad de peleas. Las peleas del pasado no interesan.

Resolución — en teoría

Ver qué tenemos que especificar, y en base a esto:

1. Definir los observadores y la igualdad observacional.
2. Definir los generadores.
3. Definir las otras operaciones.
4. Definir las restricciones donde corresponda
5. Incluir otras operaciones auxiliares, de haberlas.
6. Axiomatizar todo.

Enunciado

Insoportables es un programa televisivo muy exitoso que sale al aire todas las noches; en él se debate acerca de las relaciones entre los personajes de la farándula (los "famosos"). Con el tiempo, distintos famosos se van incorporando al programa (y nunca dejan de pertenecer al mismo).

Debido a la gran cantidad de peleas y reconciliaciones, los productores nos encargaron el desarrollo de un sistema que permita saber en todo momento quiénes están peleados y quiénes no. Además, los productores quieren poder determinar quién es el famoso que actualmente está involucrado en la mayor cantidad de peleas. Las peleas del pasado no interesan.

Enunciado

Insoportables es un programa televisivo muy exitoso que sale al aire todas las noches; en él se debate acerca de las relaciones entre los personajes de la farándula (los "famosos"). Con el tiempo, distintos famosos se van incorporando al programa (y nunca dejan de pertenecer al mismo).

Debido a la gran cantidad de peleas y reconciliaciones, los productores nos encargaron el desarrollo de un sistema que permita saber en todo momento quiénes están peleados y quiénes no. Además, los productores quieren poder determinar quién es el famoso que actualmente está involucrado en la mayor cantidad de peleas. Las peleas del pasado no interesan.

¿Qué tenemos que especificar?

¿Qué tenemos que especificar?

- El sistema debería permitir registrar nuevos famosos, nuevas peleas y nuevas reconciliaciones.
- Saber qué famosos están peleados. (¿La relación "estar peleado con" siempre es simétrica?)
- Saber quién es el famoso involucrado en la mayor cantidad de peleas. (¿Siempre hay uno?)

Definir los observadores

Definir los observadores

- Los observadores deben permitirnos distinguir todas las instancias.

Definir los observadores

- Los observadores deben permitirnos distinguir todas las instancias.
- Es decir, deberíamos poder definir todas las *otras operaciones* utilizando los observadores y sin necesidad de axiomatizar *en base a* los generadores.

Definir los observadores

- Los observadores deben permitirnos distinguir todas las instancias.
- Es decir, deberíamos poder definir todas las *otras operaciones* utilizando los observadores y sin necesidad de axiomatizar *en base a* los generadores.

famosos	: bdf	$\longrightarrow$	conj(famoso)	
enemigos	: bdf b $\times$ famoso f	$\longrightarrow$	conj(famoso)	$\{f \in \text{famosos}(b)\}$

Definir los observadores

- Los observadores deben permitirnos distinguir todas las instancias.
- Es decir, deberíamos poder definir todas las *otras operaciones* utilizando los observadores y sin necesidad de axiomatizar *en base a* los generadores.

famosos	: bdf	$\longrightarrow$	conj(famoso)	
enemigos	: bdf b $\times$ famoso f	$\longrightarrow$	conj(famoso)	$\{f \in \text{famosos}(b)\}$

- ¿Es posible responder todas las preguntas en base a esta información?

Escribir la igualdad observacional

- Ya podemos escribir la igualdad observacional.

Escribir la igualdad observacional

- Ya podemos escribir la igualdad observacional.

igualdad observacional

$$(\forall b, b' : \text{bdf}) \left(b =_{\text{obs}} b' \iff \begin{pmatrix} \text{famosos}(b) =_{\text{obs}} \text{famosos}(b') \wedge_{\text{L}} \\ (\forall f : \quad \text{famoso})(f \quad \in \quad \text{famosos}(b) \quad \Rightarrow_{\text{L}} \\ \text{enemigos}(b, f) =_{\text{obs}} \text{enemigos}(b', f)) \end{pmatrix} \right)$$

Definir los generadores

Definir los generadores

- Los generadores deben permitirnos construir todas las instancias observacionalmente distintas.

Definir los generadores

- Los generadores deben permitirnos construir todas las instancias observacionalmente distintas.

$$\begin{array}{l}
\text{crearBD} \ : \ \longrightarrow \ \text{bdf} \\
\text{nuevoFamoso} \ : \ \text{bdf } b \times \text{famoso } f \ \longrightarrow \ \text{bdf} \qquad \{f \notin \text{famosos}(b)\} \\
\text{pelear} \ : \ \text{bdf } b \times \text{famoso } f \times \text{famoso } f' \ \longrightarrow \ \text{bdf} \\
\qquad \{\{f, f'\} \subseteq \text{famosos}(b) \wedge_{\text{L}} f \notin \text{enemigos}(b, f') \wedge f \neq f'\}
\end{array}$$

Definir los generadores

- Los generadores deben permitirnos construir todas las instancias observacionalmente distintas.

crearBD : $\longrightarrow$ bdf	
nuevoFamoso : bdf $b \times$ famoso f $\longrightarrow$ bdf	$\{f \notin \text{famosos}(b)\}$
pelear : bdf $b \times$ famoso $f \times$ famoso f' $\longrightarrow$ bdf	$\{\{f, f'\} \subseteq \text{famosos}(b) \wedge_{\text{L}} f \notin \text{enemigos}(b, f') \wedge f \neq f'\}$

- ¿Podemos generar todas las instancias?

Definir las otras operaciones

Definir las otras operaciones

- Las otras operaciones tienen que ser suficientes para permitir utilizar el TAD fácilmente.

Definir las otras operaciones

- Las otras operaciones tienen que ser suficientes para permitir utilizar el TAD fácilmente.

reconciliar	:	bdf $b \times$ famoso $f \times$ famoso f'	$\longrightarrow$	bdf $\{\{f, f'\} \subseteq \text{famosos}(b) \wedge_{\text{L}} (f \in \text{enemigos}(b, f'))\}$
másPeleador	:	bdf b	$\longrightarrow$	famoso $\{\text{famosos}(b) \neq \emptyset\}$

Definir las otras operaciones

- Las otras operaciones tienen que ser suficientes para permitir utilizar el TAD fácilmente.

reconciliar	:	bdf $b \times$ famoso $f \times$ famoso f' $\longrightarrow$ bdf	$\{\{f, f'\} \subseteq \text{famosos}(b) \wedge_{\text{L}} (f \in \text{enemigos}(b, f'))\}$
másPeleador	:	bdf b $\longrightarrow$ famoso	$\{\text{famosos}(b) \neq \emptyset\}$

- ¿Se pueden definir solamente en base a los observadores?

Axiomatización I

- Encuentre el/los error/es:

enemigos : bdf b $\times$ famoso f $\longrightarrow$ conj(famoso) $\{f \in \text{famosos}(b)\}$

enemigos(crearBD, f) $\equiv$ $\emptyset$

enemigos(nuevoFamoso(b, g), f) $\equiv$ enemigos(b, f)

enemigos(pelear(b, g, g'), f) $\equiv$ **if** $f \in \{g, g'\}$ **then**
$\{g, g'\} \setminus \{f\}$
else
$\emptyset$
fi $\cup$ enemigos(b, f)

Axiomatización I

- Encuentre el/los error/es:

enemigos : bdf b × famoso f $\longrightarrow$ conj(famoso) $\{f \in \text{famosos}(b)\}$

enemigos(crearBD, f) $\equiv$ $\emptyset$

enemigos(nuevoFamoso(b, g), f) $\equiv$ enemigos(b, f)

enemigos(pelear(b, g, g'), f) $\equiv$ **if** $f \in \{g, g'\}$ **then**
$\{g, g'\} \setminus \{f\}$
else
$\emptyset$
fi $\cup$ enemigos(b, f)

- ¿Qué pasa con las restricciones?

Axiomatización I

Axiomatización I

$$\begin{array}{lll}
\text{enemigos} : \text{bdf } b \times \text{famoso } f \longrightarrow \text{conj(famoso)} & & \{f \in \text{famosos}(b)\} \\
\\
\text{enemigos}(\text{nuevoFamoso}(b, g), f) & \equiv & \textbf{if } g = f \textbf{ then} \\
& & \quad \emptyset \\
& & \textbf{else} \\
& & \quad \text{enemigos}(b, f) \\
& & \textbf{fi} \\
\\
\text{enemigos}(\text{pelear}(b, g, g'), f) & \equiv & \textbf{if } f \in \{g, g'\} \textbf{ then} \\
& & \quad \{g, g'\} \setminus \{f\} \\
& & \textbf{else} \\
& & \quad \emptyset \\
& & \textbf{fi} \cup \text{enemigos}(b, f)
\end{array}$$

Axiomatización I

- ¿Qué hay de raro acá?

reconciliar : bdf b $\times$ famoso f $\times$ famoso f' $\longrightarrow$ bdf
$\{\{f, f'\} \subseteq \text{famosos}(b) \wedge f \in \text{enemigos}(b, f')\}$

reconciliar(pelear(b, g, g'), f, f') $\equiv$ **if** $\{g, g'\} = \{f, f'\}$ **then**
    b
else
    pelear(reconciliar(b, f, f'), g, g')
fi

reconciliar(nuevoFamoso(b, g), f, f') $\equiv$ **if** $g \in \{f, f'\}$ **then**
    b
else
    nuevoFamoso(reconciliar(b, f, f'), g)
fi

Axiomatización I

- ¿Qué hay de raro acá?

reconciliar : bdf $b \times$ famoso $f \times$ famoso $f' \longrightarrow$ bdf
$\{\{f, f'\} \subseteq \text{famosos}(b) \wedge f \in \text{enemigos}(b, f')\}$

reconciliar(pelear(b, g, g'), f, f') $\equiv$ **if** $\{g, g'\} = \{f, f'\}$ **then**
    b
else
    pelear(reconciliar(b, f, f'), g, g')
fi

reconciliar(nuevoFamoso(b, g), f, f') $\equiv$ **if** $g \in \{f, f'\}$ **then**
    b
else
    nuevoFamoso(reconciliar(b, f, f'), g)
fi

- ¿Es incorrecto chequear que $g \in \{f, f'\}$?

Axiomatización I

- ¿Qué hay de raro acá?

reconciliar : bdf $b \times$ famoso $f \times$ famoso $f' \longrightarrow$ bdf
$\{\{f, f'\} \subseteq \text{famosos}(b) \wedge f \in \text{enemigos}(b, f')\}$

reconciliar(pelear(b, g, g'), f, f') $\equiv$ **if** $\{g, g'\} = \{f, f'\}$ **then**
    b
else
    pelear(reconciliar(b, f, f'), g, g')
fi

reconciliar(nuevoFamoso(b, g), f, f') $\equiv$ **if** $g \in \{f, f'\}$ **then**
    b
else
    nuevoFamoso(reconciliar(b, f, f'), g)
fi

- ¿Es incorrecto chequear que $g \in \{f, f'\}$?
- ¿Qué pasa con las restricciones?

Axiomatización I

reconciliar : bdf b $\times$ famoso f $\times$ famoso f' $\longrightarrow$ bdf
$\{\{f, f'\} \subseteq \text{famosos}(b) \wedge f \in \text{enemigos}(b, f')\}$

reconciliar(pelear(b, g, g'), f, f') $\equiv$ **if** $\{g, g'\} = \{f, f'\}$ **then**
b
else
pelear(reconciliar(b, f, f'), g, g')
fi

reconciliar(nuevoFamoso(b, g), f, f') $\equiv$ nuevoFamoso(reconciliar(b, f, f'), g)

Axiomatización II

- Encuentre el/los posible/s error/es.

$\text{másPeleador}(b) \equiv \text{prim}(\text{másPeleadores}(b))$
donde
$\text{másPeleadores} : \text{bdf} \longrightarrow \text{secu}(\text{famoso})$

Axiomatización II

- Encuentre el/los posible/s error/es.

$\text{másPeleador}(b) \equiv \text{prim}(\text{másPeleadores}(b))$
donde
$\text{másPeleadores} : \text{bdf} \longrightarrow \text{secu}(\text{famoso})$

- ¿Qué hay de raro acá?

Axiomatización II

- Encuentre el/los posible/s error/es.

$\text{másPeleador}(b) \equiv \text{prim}(\text{másPeleadores}(b))$
donde
$\text{másPeleadores} : \text{bdf} \longrightarrow \text{secu}(\text{famoso})$

- ¿Qué hay de raro acá?
- ¿Qué pasa con las instancias observacionalmente iguales pero que están construidas de diferente manera?

Axiomatización II: congruencia

Definición

Una relación de equivalencia $\equiv$ en un conjunto X es una congruencia con respecto a una función $f : X \to X$ si cada vez que $x \equiv y$ se tiene que $f(x) \equiv f(y)$.

- En particular, la igualdad observacional $=_{obs}$ es una congruencia con respecto a una operación f si y sólo si para cada par $i =_{obs} i'$ también vale $f(i) =_{obs} f(i')$.

Axiomatización II: congruencia

Definición

Una relación de equivalencia $\equiv$ en un conjunto X es una congruencia con respecto a una función $f : X \to X$ si cada vez que $x \equiv y$ se tiene que $f(x) \equiv f(y)$.

- En particular, la igualdad observacional $=_{obs}$ es una congruencia con respecto a una operación f si y sólo si para cada par $i =_{obs} i'$ también vale $f(i) =_{obs} f(i')$.
- Una solución correcta:

$$\text{másPeleador}(b) \equiv \text{dameUno}(\text{másPeleadores}(b))$$
donde
$$\text{másPeleadores} : \text{bdf} \longrightarrow \text{conj(famoso)}$$

Minimalidad

- ¿Sería buena idea agregar el siguiente observador?

sonEnemigos? : bdf b $\times$ famoso f $\times$ famoso f' $\longrightarrow$ bool	$\{\ldots\}$

Minimalidad

- ¿Sería buena idea agregar el siguiente observador?

sonEnemigos? : bdf b $\times$ famoso f $\times$ famoso f' $\longrightarrow$ bool	$\{\ldots\}$

- ¿Sería buena idea que "reconciliar" fuese un generador?

reconciliar : bdf b $\times$ famoso f $\times$ famoso f' $\longrightarrow$ bdf	$\{\ldots\}$

Minimalidad

- ¿Sería buena idea agregar el siguiente observador?

sonEnemigos? : bdf b $\times$ famoso f $\times$ famoso f' $\longrightarrow$ bool	$\{\ldots\}$

- ¿Sería buena idea que "reconciliar" fuese un generador?

reconciliar : bdf b $\times$ famoso f $\times$ famoso f' $\longrightarrow$ bdf	$\{\ldots\}$

En el contexto de Algo 2:

Minimalidad

- ¿Sería buena idea agregar el siguiente observador?

sonEnemigos? : bdf b × famoso f × famoso f' ⟶ bool	{...}

- ¿Sería buena idea que "reconciliar" fuese un generador?

reconciliar : bdf b × famoso f × famoso f' ⟶ bdf	{...}

En el contexto de Algo 2:

- El conjunto de observadores *debe* ser minimal.
- Muchas veces es *deseable* que el conjunto de generadores sea minimal, pero depende del problema que estemos resolviendo.

Conclusiones

- Proceso de construcción de un TAD.
  - Determinar qué hay que especificar.
  - Observadores básicos e igualdad observacional
  - Generadores
  - Otras operaciones
  - Axiomas
  - Operaciones auxiliares

Conclusiones

- Proceso de construcción de un TAD.
  - Determinar qué hay que especificar.
  - Observadores básicos e igualdad observacional
  - Generadores
  - Otras operaciones
  - Axiomas
  - Operaciones auxiliares

- ¿Es posible escribir por completo los generadores antes de ponerse a pensar siquiera en los observadores?

Conclusiones

- Proceso de construcción de un TAD.
  - Determinar qué hay que especificar.
  - Observadores básicos e igualdad observacional
  - Generadores
  - Otras operaciones
  - Axiomas
  - Operaciones auxiliares
- ¿Es posible escribir por completo los generadores antes de ponerse a pensar siquiera en los observadores?
- ¿Es posible escribir por completo los observadores antes de ponerse a pensar siquiera en los generadores?

Conclusiones

- Proceso de construcción de un TAD.
  - Determinar qué hay que especificar.
  - Observadores básicos e igualdad observacional
  - Generadores
  - Otras operaciones
  - Axiomas
  - Operaciones auxiliares
- ¿Es posible escribir por completo los generadores antes de ponerse a pensar siquiera en los observadores?
- ¿Es posible escribir por completo los observadores antes de ponerse a pensar siquiera en los generadores?
- Usualmente todo esto termina siendo un proceso iterativo.