Clase práctica P01: Introducción a TADs

Parte 1: Repaso de recursión

Algoritmos y Estructuras de Datos II

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

2do cuatrimestre de 2020 (a distancia)

¿Qué es la recursión?

- ¿Qué es una función recursiva?

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

- En general, una función recursiva tiene,
  - uno o más *casos base*
  - uno o más llamados 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" a algún caso base.

¿Cómo escribir funciones recursivas?

Ejemplo

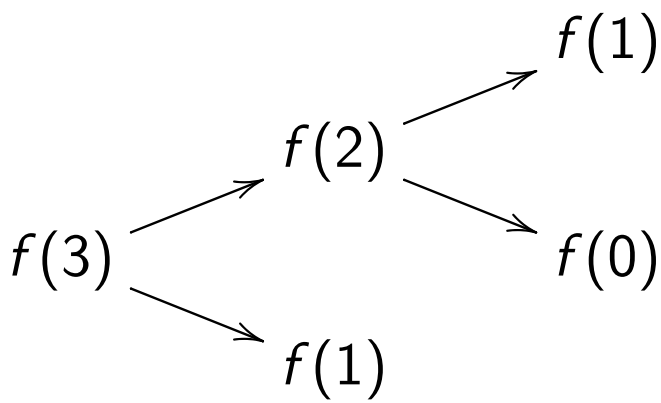
$$\mathsf{f} : \mathsf{nat} \longrightarrow \mathsf{nat}$$

$$\begin{array}{l} \mathsf{f}(n) \equiv \mathbf{if}\ n = 0\ \mathbf{then} \\ \qquad\quad 0 \\ \qquad \mathbf{else} \\ \qquad\quad (\mathbf{if}\ n = 1\ \mathbf{then}\ 2\ \mathbf{else}\ \mathsf{f}(n-1) + \mathsf{f}(n-2)\ \mathbf{fi}) \\ \qquad \mathbf{fi} \end{array}$$

$$f(3) \equiv f(2) + f(1) \equiv (f(1) + f(0)) + f(1) \equiv (2 + 0) + 2 \equiv 4$$

¿Cómo escribir funciones recursivas?

$$
\begin{array}{l}
\mathsf{f} : \mathsf{nat} \longrightarrow \mathsf{nat} \\
\mathsf{f}(n) \equiv \mathbf{if}\ n = 0\ \mathbf{then} \\
\qquad\qquad 0 \\
\qquad \mathbf{else} \\
\qquad\qquad (\mathbf{if}\ n = 1\ \mathbf{then}\ 2\ \mathbf{else}\ \mathsf{f}(n-1) + \mathsf{f}(n-2)\ \mathbf{fi}) \\
\qquad \mathbf{fi}
\end{array}
$$



$3 > 2 > 1$
$3 > 2 > 0$
$3 > 1$

¿Cómo escribir funciones recursivas?

Más en general, si $>$ es una relación binaria sobre un conjunto X.

Definición (repaso de Álgebra I)

La relación $>$ es una relación de orden[1] si es:

1. Irreflexiva $\forall x \in X.\ \neg(x > x)$
2. Asimétrica $\forall x, y \in X.\ (x > y) \Rightarrow \neg(y > x)$
3. Transitiva $\forall x, y, z \in X.\ (x > y \wedge y > z) \Rightarrow x > z$

Definición

Además, $>$ es una relación de orden bien fundado si no existe ninguna sucesión infinita de elementos $x_1, x_2, x_3, \ldots \in X$ tal que:

$$x_1 > x_2 > x_3 > \ldots$$

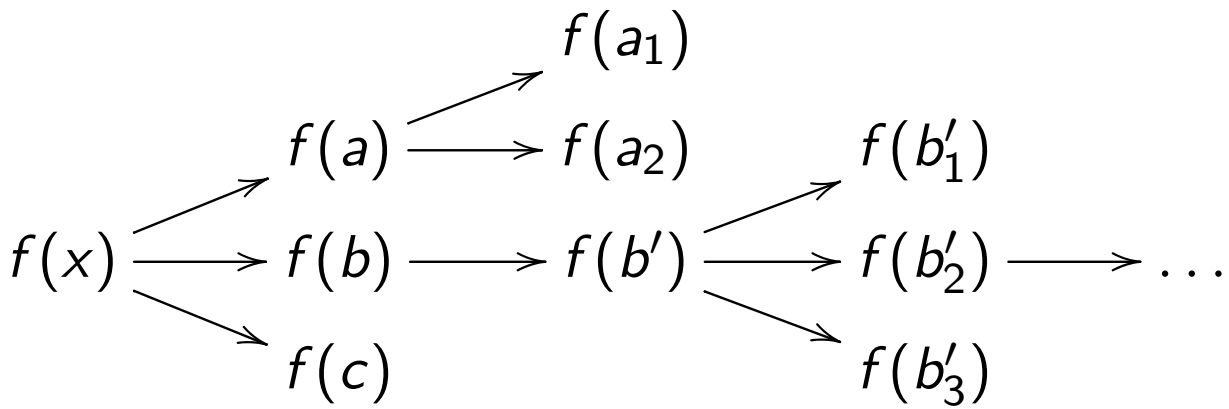
[1] Más precisamente, un orden parcial estricto.

¿Cómo escribir funciones recursivas?

Teorema

Una función $f : X \to Y$ recursiva queda bien definida, es decir, termina, si se puede encontrar un orden bien fundado $>$ sobre X, de tal manera que si el valor $f(x)$ se computa a partir de los valores de $f(x_1), \ldots, f(x_n)$ entonces $x > x_i$ para todo $1 \leq i \leq n$.

Gráficamente



¿Cómo escribir funciones recursivas?

¿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?

¿Cómo escribir funciones recursivas?

Ejemplo

$$\begin{aligned}
&\mathsf{f} : \mathsf{nat} \longrightarrow \mathsf{nat} \\
&\mathsf{f}(n) \equiv \mathbf{if}\ n = 0\ \mathbf{then} \\
&\qquad 0 \\
&\qquad \mathbf{else} \\
&\qquad\quad (\mathbf{if}\ n = 1\ \mathbf{then}\ 2\ \mathbf{else}\ \mathsf{f}(n-1) + \mathsf{f}(n-2)\ \mathbf{fi}) \\
&\qquad \mathbf{fi}
\end{aligned}$$

Lema. Para todo n : nat, se tiene que $f(n)$ es par.

Demostración.

1. **Terminación.** En los llamados recursivos $f(n)$ se calcula en función de $f(n-1)$ y $f(n-2)$ donde $n > n-1$ y $n > n-2$. Además, el orden $>$ de los naturales es bien fundado.
2. **Casos base.** Notar que $f(0) \equiv 0$ es par y $f(1) \equiv 2$ es par.
3. **Casos recursivos.** Si $n > 1$, tenemos que $f(n) \equiv f(n-1) + f(n-2)$. Asumiendo que $f(n-1)$ y $f(n-2)$ son pares, concluímos que $f(n)$ es par.

Clase práctica P01: Introducción a TADs

Parte 2: Apunte de TADs básicos

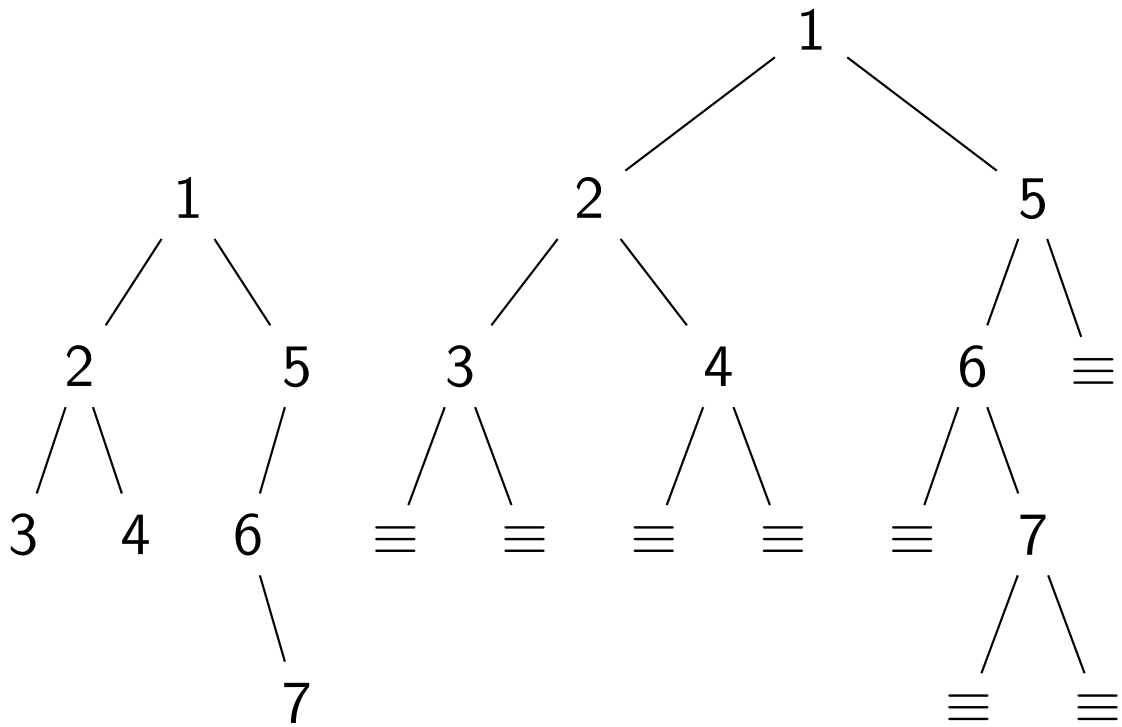
Algoritmos y Estructuras de Datos II

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

2do cuatrimestre de 2020 (a distancia)

¿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.


1
1
2
5
2
5
3
4
6
≡
3
4
6
≡
≡
≡
≡
≡
≡
7
7
≡
≡

Clase práctica P01: Introducción a TADs

Parte 3: Ejercicios de recursión

Algoritmos y Estructuras de Datos II

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

2do cuatrimestre de 2020 (a distancia)

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.

$\text{esPrefijo?} \ : \ \text{secu}(\alpha) \times \text{secu}(\alpha) \ \longrightarrow \ \text{bool}$ $\{\ \}$

$\text{esPrefijo?}(<>, t) \ \equiv \ \mathit{true}$

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

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*.

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

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

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

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*.

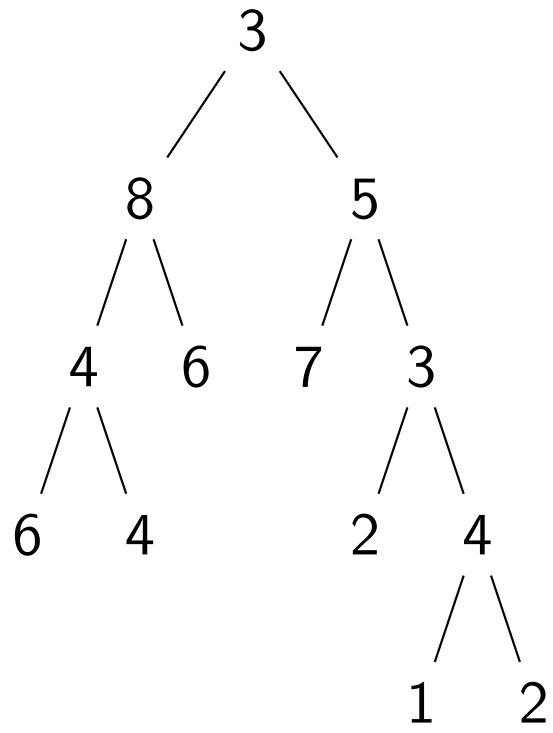
$\#\text{Apariciones} : \text{ab}(\alpha) \times \alpha \longrightarrow \text{nat}$ $\{\ \}$

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

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

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).



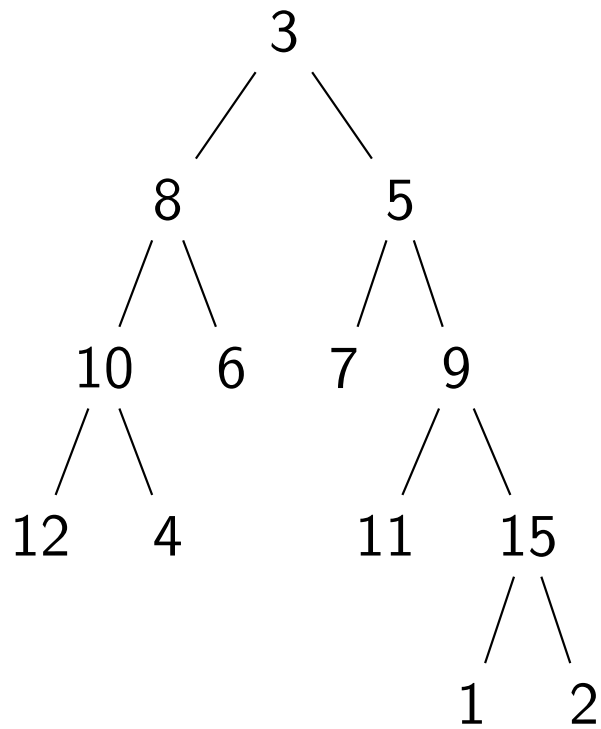
últimoNivelCompleto : ab(α) $\longrightarrow$ nat

últimoNivelCompleto(*nil*) $\equiv$ 0

{ }

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*) $\equiv$ <>

Clase práctica P01: Introducción a TADs

Parte 4: Ejercicio — Agenda de compromisos

Algoritmos y Estructuras de Datos II

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

2do cuatrimestre de 2020 (a distancia)

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, quisiéramos poder saber qué hora del intervalo es la más ocupada en la agenda.

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, quisiéramos poder saber qué hora del intervalo es la más ocupada en la agenda.

- 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 hora de un intervalo es la más ocupada.

Observadores, generadores e igualdad

- ¿Observadores?

$$\begin{array}{l} \text{compromisos} : \text{agenda} \times \text{nat} \longrightarrow \text{conj(compromiso)} \\ \text{horaMasOcupada} : \text{agenda} \times \text{nat} \times \text{nat} \longrightarrow \text{nat} \end{array}$$

- ¿Igualdad observacional?

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

- ¿Generadores?

$$\begin{array}{l} \text{crearAgenda} : \longrightarrow \text{agenda} \\ \text{registrar} : \text{agenda} \times \text{compromiso} \times \text{nat} \times \text{nat} \longrightarrow \text{agenda} \\ \text{registrar} : \text{agenda} \times \text{compromiso} \times \text{nat}\ d \times \text{nat}\ h \longrightarrow \text{agenda} \quad \{d < h\} \end{array}$$

- ¿Otras operaciones?

$$\begin{array}{l} \text{horaMasOcupada} : \text{agenda} \times \text{nat} \times \text{nat} \longrightarrow \text{nat} \\ \text{horaMasOcupada} : \text{agenda} \times \text{nat}\ d \times \text{nat}\ h \longrightarrow \text{nat} \quad \{d \leq h\} \end{array}$$

Axiomas

compromisos : agenda $\times$ nat $\longrightarrow$ conj(compromiso)

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

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

horaMasOcupada : agenda $\times$ nat d $\times$ nat h $\longrightarrow$ nat $\{d \leq h\}$

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

Clase práctica P01: Introducción a TADs

Parte 5: Ejercicio — Insoportables (modelado)

Algoritmos y Estructuras de Datos II

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

2do cuatrimestre de 2020 (a distancia)

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.

¿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

- Los observadores deben permitirnos distinguir todas las instancias.
- Es decir, deberíamos poder definir todas las operaciones a partir de los observadores.

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.

igualdad observacional

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

Definir los generadores

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

- 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 y aplicación de generadores?

Clase práctica P01: Introducción a TADs

Parte 6: Ejercicio — Insoportables (axiomatización)

Algoritmos y Estructuras de Datos II

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

2do cuatrimestre de 2020 (a distancia)

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

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

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

- ¿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
$\text{másPeleadores} : \text{bdf} \longrightarrow \text{secu}(\text{famoso})$

- ¿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 vale $x \equiv y$ 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
$\text{másPeleadores} : \text{bdf} \longrightarrow \text{conj(famoso)}$

- Otra solución:

$\text{másPeleador}(b) \equiv \text{elegirUnMásPeleador}(b, \text{famosos}(b))$
donde
$\text{elegirUnMásPeleador} : \text{bdf } b \times \text{conj(famoso) } cf \longrightarrow \text{famoso}$
$\{\neg\emptyset(cf) \wedge cf \subseteq \text{famosos}(b)\}$

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:

- El conjunto de observadores debe ser minimal.
- Es *deseable* que el conjunto de generadores sea minimal.
- De lo contrario, la especificación se torna más difícil, menos clara y más propensa a errores.

Enunciado (bis)

Supongamos que los productores de *Insoportables* están contentos con la especificación entregada, pero que ahora también quieren poder determinar cuáles son los famosos que más se pelearon en su vida.

¿Qué modificaciones hay que hacerle al TAD?

¿Qué tenemos que especificar? (bis)

- El sistema debería permitir registrar nuevos famosos, nuevas peleas y nuevas reconciliaciones.
- Saber qué famosos están peleados.
- Saber quién es el famoso involucrado en la mayor cantidad de peleas.
- Quiénes son los famosos que más veces se pelearon en su vida.

Definir los generadores (bis)

- ¿Podemos construir todas las instancias observacionalmente distintas con los generadores que tenemos? (crearBD, nuevoFamoso, pelear)
- ¡Ya no, porque ahora (parte de) la historia es observable!
- Solución: agregar "reconciliar" como un generador.

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

- ¿Podemos generar ahora todas las instancias?
- Con esta función, tenemos memoria de todas las peleas históricas.
- Ahora que "reconciliar" es un generador, no se axiomatiza, pero hay que axiomatizar las funciones para el nuevo generador (cuando corresponda).

Definir los observadores (bis)

- Necesitamos una operación que nos permita determinar quiénes son los más peleadores históricos.

másPeleadoresHistóricos : bdf b $\longrightarrow$ conj(famoso)

- ¿Estaría bien incluir esta operación como otra operación?
- Cuidado con la congruencia.
- ¿Estaría bien incluir esta operación como observador?
- Cuidado con la congruencia (más sutil; pensarlo).

Definir los observadores (bis)

- Una forma de resolver esto correctamente sería agregar como observador básico una función que permita determinar la cantidad de peleas (incluyendo historia) de un famoso.

#peleasHistórico : bdf b × famoso f $\longrightarrow$ nat	$\{f \in \text{famosos}(b)\}$

- Notar que este observador capturaría un *poco* más de detalle del que el enunciado pedía.
- Si tuviéramos un observador que devuelva el historial completo de peleas de un famoso f dado, también podríamos usarlo para calcular #peleasHistórico. Sin embargo, esto capturaría **mucho** más detalle del que el enunciado pedía (¡y distinguiría instancias que no nos pidieron distinguir!).
- ... esto sería **sobreespecificar**.

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.