Matemáticas Discretas I

Definiciones recursivas e Inducción Estructural

Juan Francisco Díaz Frias

Profesor Titular (1993-hoy)
juanfco.diaz@correounivalle.edu.co
Edif. 331 - 2111
Universidad del Valle

Noviembre 2018

Plan

Plan

Plan

Plan

Motivación

- Algunas veces, es difícil definir un objeto explícitamente. Y es más fácil hacerlo en términos de sí mismo: definiciones recursivas.
- Podemos definir recursivamente funciones y conjuntos. La definición recursiva de conjuntos da lugar a estructuras, cuyas propiedades se prueban usando una especie de inducción llamada inducción estructural.
- Las secuencias, los árboles, las cadenas de caracteres, las fórmulas de la lógica proposicional y del cálculo de predicados son ejemplos de conjuntos muy usados en informática, definidos recursivamente.
- Vamos a estudiar con detenimiento algunos de estos objetos definidos recursivamente y cómo probar propiedades sobre ellos.

Motivación

- Algunas veces, es difícil definir un objeto explícitamente. Y es más fácil hacerlo en términos de sí mismo: definiciones recursivas.
- Podemos definir recursivamente funciones y conjuntos. La definición recursiva de conjuntos da lugar a estructuras, cuyas propiedades se prueban usando una especie de inducción llamada inducción estructural.
- Las secuencias, los árboles, las cadenas de caracteres, las fórmulas de la lógica proposicional y del cálculo de predicados son ejemplos de conjuntos muy usados en informática, definidos recursivamente.
- Vamos a estudiar con detenimiento algunos de estos objetos definidos recursivamente y cómo probar propiedades sobre ellos.

Motivación

- Algunas veces, es difícil definir un objeto explícitamente. Y es más fácil hacerlo en términos de sí mismo: definiciones recursivas.
- Podemos definir recursivamente funciones y conjuntos. La definición recursiva de conjuntos da lugar a estructuras, cuyas propiedades se prueban usando una especie de inducción llamada inducción estructural.
- Las secuencias, los árboles, las cadenas de caracteres, las fórmulas de la lógica proposicional y del cálculo de predicados son ejemplos de conjuntos muy usados en informática, definidos recursivamente.
- Vamos a estudiar con detenimiento algunos de estos objetos definidos recursivamente y cómo probar propiedades sobre ellos.

Motivación

- Algunas veces, es difícil definir un objeto explícitamente. Y es más fácil hacerlo en términos de sí mismo: definiciones recursivas.
- Podemos definir recursivamente funciones y conjuntos. La definición recursiva de conjuntos da lugar a estructuras, cuyas propiedades se prueban usando una especie de inducción llamada inducción estructural.
- Las secuencias, los árboles, las cadenas de caracteres, las fórmulas de la lógica proposicional y del cálculo de predicados son ejemplos de conjuntos muy usados en informática, definidos recursivamente.
- Vamos a estudiar con detenimiento algunos de estos objetos definidos recursivamente y cómo probar propiedades sobre ellos.

Plan

1. Motivación
2. Definiciones recursivas
   - . . . de Funciones
   - . . . de Conjuntos
3. Las secuencias y los árboles como estructuras discretas
   - ¿Qué entendemos por *estructuras*?
   - Las secuencias
   - Los árboles binarios
4. Inducción estructural
   - Inducción estructural
   - Teoremas sobre secuencias
   - Teoremas sobre árboles

Definición recursiva de funciones (1)

Números de Fibonacci (Video 3.8.1)

Una pareja de conejos recién nacidos (macho y hembra) es dejada en una isla. Se sabe que los conejos no se pueden reproducir hasta que no cumplen dos meses de edad. Una vez una pareja de conejos cumple 2 meses de edad, se reproducen y dan a luz una pareja nueva. Se quiere definir una función $fib : \mathbb{N} \to \mathbb{N}$ tal que $fib(n)$ represente el número de parejas de conejos en la isla después de pasados n meses. Suponga que ninguna pareja de conejos se muere.

Miremos el comportamiento en una tabla:

Mes	Edad			Total
	≥ 2	1	0	
0	0	0	1	1
1	0	1	0	1
2	1	0	1	2
3	1	1	1	3
4	2	1	2	5
5	3	2	3	8
6	5	3	5	13

Escrito recursivamente:

$$fib(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ fib(n-2) + fib(n-1) & n \geq 2 \end{cases}$$

Definición recursiva de funciones (1)

Números de Fibonacci (Video 3.8.1)

Una pareja de conejos recién nacidos (macho y hembra) es dejada en una isla. Se sabe que los conejos no se pueden reproducir hasta que no cumplen dos meses de edad. Una vez una pareja de conejos cumple 2 meses de edad, se reproducen y dan a luz una pareja nueva. Se quiere definir una función $fib : \mathbb{N} \rightarrow \mathbb{N}$ tal que $fib(n)$ represente el número de parejas de conejos en la isla después de pasados n meses. Suponga que ninguna pareja de conejos se muere.

Miremos el comportamiento en una tabla:

Mes	Edad			Total
	≥ 2	1	0	
0	0	0	1	1
1	0	1	0	1
2	1	0	1	2
3	1	1	1	3
4	2	1	2	5
5	3	2	3	8
6	5	3	5	13

Escrito recursivamente:

$$fib(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ fib(n-2) + fib(n-1) & n \geq 2 \end{cases}$$

Definición recursiva de funciones (1)

Números de Fibonacci (Video 3.8.1)

Una pareja de conejos recién nacidos (macho y hembra) es dejada en una isla. Se sabe que los conejos no se pueden reproducir hasta que no cumplen dos meses de edad. Una vez una pareja de conejos cumple 2 meses de edad, se reproducen y dan a luz una pareja nueva. Se quiere definir una función $fib : \mathbb{N} \to \mathbb{N}$ tal que $fib(n)$ represente el número de parejas de conejos en la isla después de pasados n meses. Suponga que ninguna pareja de conejos se muere.

Miremos el comportamiento en una tabla:

Mes	Edad			Total
	≥ 2	1	0	
0	0	0	1	1
1	0	1	0	1
2	1	0	1	2
3	1	1	1	3
4	2	1	2	5
5	3	2	3	8
6	5	3	5	13

Escrito recursivamente:

$$fib(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ fib(n-2) + fib(n-1) & n \geq 2 \end{cases}$$

Definición recursiva de funciones (2)

Torres de Hanoi (Video 3.8.2)

Dios, al crear el mundo, colocó tres varillas de diamante con 64 discos apilados de mayor a menor radio en la primera varilla. También creó un monasterio con monjes, quienes tenían la tarea de resolver esta Torre de Hanoi divina. Para poder resolverla, los monjes debían llevar todos los discos a la última varilla quedando en el mismo orden a como estaban en la primera; además, se impuso tres condiciones: sólo se puede mover un disco a la vez, un disco de mayor diámetro no puede descansar sobre otro diámetro menor y sólo se puede mover el disco que se encuentra arriba en cada varilla. Dios profetizó que el día en que estos monjes consiguieran terminar el juego, el mundo acabaría. Si los monjes movieran cada segundo un disco, sin equivocarse, y la historia fuera cierta, ¿Podrías decir cuándo será el fin del mundo?

Para responder la pregunta, se quiere definir una función $hanoi : \mathbb{N} \rightarrow \mathbb{N}$ tal que $hanoi(n)$ represente el número de movimientos mínimo necesario para mover una torre de n discos.

Escrito recursivamente:

Juguemos con las torres de Hanoi

$$hanoi(n) = \begin{cases} 1 & n = 1 \\ 2hanoi(n-1) + 1 & n \geq 2 \end{cases}$$

Definición recursiva de funciones (2)

Torres de Hanoi (Video 3.8.2)

Dios, al crear el mundo, colocó tres varillas de diamante con 64 discos apilados de mayor a menor radio en la primera varilla. También creó un monasterio con monjes, quienes tenían la tarea de resolver esta Torre de Hanoi divina. Para poder resolverla, los monjes debían llevar todos los discos a la última varilla quedando en el mismo orden a como estaban en la primera; además, se impuso tres condiciones: sólo se puede mover un disco a la vez, un disco de mayor diámetro no puede descansar sobre otro diámetro menor y sólo se puede mover el disco que se encuentra arriba en cada varilla. Dios profetizó que el día en que estos monjes consiguieran terminar el juego, el mundo acabaría. Si los monjes movieran cada segundo un disco, sin equivocarse, y la historia fuera cierta, ¿Podrías decir cuándo será el fin del mundo?
Para responder la pregunta, se quiere definir una función $hanoi : \mathbb{N} \rightarrow \mathbb{N}$ tal que $hanoi(n)$ represente el número de movimientos mínimo necesario para mover una torre de n discos.

Escrito recursivamente:

Juguemos con las torres de Hanoi

$$hanoi(n) = \begin{cases} 1 & n = 1 \\ 2hanoi(n-1)+1 & n \geq 2 \end{cases}$$

Definición recursiva de funciones (2)

Torres de Hanoi (Video 3.8.2)

Dios, al crear el mundo, colocó tres varillas de diamante con 64 discos apilados de mayor a menor radio en la primera varilla. También creó un monasterio con monjes, quienes tenían la tarea de resolver esta Torre de Hanoi divina. Para poder resolverla, los monjes debían llevar todos los discos a la última varilla quedando en el mismo orden a como estaban en la primera; además, se impuso tres condiciones: sólo se puede mover un disco a la vez, un disco de mayor diámetro no puede descansar sobre otro diámetro menor y sólo se puede mover el disco que se encuentra arriba en cada varilla. Dios profetizó que el día en que estos monjes consiguieran terminar el juego, el mundo acabaría. Si los monjes movieran cada segundo un disco, sin equivocarse, y la historia fuera cierta, ¿Podrías decir cuándo será el fin del mundo?
Para responder la pregunta, se quiere definir una función $hanoi : \mathbb{N} \rightarrow \mathbb{N}$ tal que $hanoi(n)$ represente el número de movimientos mínimo necesario para mover una torre de n discos.

Escrito recursivamente:

Juguemos con las torres de Hanoi

$$hanoi(n) = \begin{cases} 1 & n = 1 \\ 2hanoi(n-1) + 1 & n \geq 2 \end{cases}$$

Definición recursiva de funciones (2)

Torres de Hanoi (Video 3.8.2)

Dios, al crear el mundo, colocó tres varillas de diamante con 64 discos apilados de mayor a menor radio en la primera varilla. También creó un monasterio con monjes, quienes tenían la tarea de resolver esta Torre de Hanoi divina. Para poder resolverla, los monjes debían llevar todos los discos a la última varilla quedando en el mismo orden a como estaban en la primera; además, se impuso tres condiciones: sólo se puede mover un disco a la vez, un disco de mayor diámetro no puede descansar sobre otro diámetro menor y sólo se puede mover el disco que se encuentra arriba en cada varilla. Dios profetizó que el día en que estos monjes consiguieran terminar el juego, el mundo acabaría. Si los monjes movieran cada segundo un disco, sin equivocarse, y la historia fuera cierta, ¿Podrías decir cuándo será el fin del mundo?
Para responder la pregunta, se quiere definir una función $hanoi : \mathbb{N} \to \mathbb{N}$ tal que $hanoi(n)$ represente el número de movimientos mínimo necesario para mover una torre de n discos.

Escrito recursivamente:

Juguemos con las torres de Hanoi

$$hanoi(n) = \begin{cases} 1 & n = 1 \\ 2hanoi(n-1) + 1 & n \geq 2 \end{cases}$$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.
- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.
- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.
- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.
- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.
- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \land n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \land P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \land P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.
- Por tanto, $\forall n | n \in \mathbb{N} \land n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.
- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.

Tes ($P(k+1)$):	$fib(k+1) > (k+1)$	
Hip. Ind. ($(\forall j \| 4 \leq j \leq k : P(j))$):	$HI : (\forall j \| 4 \leq j \leq k : fib(j) > j)$	
	Exp.	Just.
	$fib(k+1)$	
$=$	$fib(k-1) + fib(k)$	Def. de *fib*
$>$	$(k-1) + k$	HI
$>$	$k+1$	$(k-1) > 1$ para $k \geq 4$
		✓

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.

Teo ($P(k+1)$):	$fib(k+1) > (k+1)$	
Hip. Ind. ($(\forall j \| 4 \leq j \leq k : P(j))$):	$HI : (\forall j \| 4 \leq j \leq k : fib(j) > j)$	
	Exp.	Just.
	$fib(k+1)$	
$=$	$fib(k-1) + fib(k)$	**Def. de** *fib*
$>$	$(k-1) + k$	**HI**
$>$	$k+1$	$(k-1) > 1$ **pues** $k \geq 5$
		$\diamond$

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.

Teo ($P(k+1)$):	$fib(k+1) > (k+1)$	
Hip. Ind. ($(\forall j \| 4 \leq j \leq k : P(j))$):	$HI : (\forall j \| 4 \leq j \leq k : fib(j) > j)$	
	Exp.	Just.
	$fib(k+1)$	
$=$	$fib(k-1) + fib(k)$	**Def. de** *fib*
$>$	$(k-1) + k$	**HI**
$>$	$k+1$	$(k-1) > 1$ **pues** $k \geq 5$
		$\diamond$

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.

Teo $(P(k+1))$**:**	$fib(k+1) > (k+1)$	
Hip. Ind. $((\forall j | 4 \leq j \leq k : P(j)))$**:**	$HI : (\forall j | 4 \leq j \leq k : fib(j) > j)$	
	Exp.	Just.
	$fib(k+1)$	
$=$	$fib(k-1) + fib(k)$	**Def. de** *fib*
$>$	$(k-1) + k$	**HI**
$>$	$k+1$	$(k-1) > 1$ **pues** $k \geq 5$
		$\diamond$

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *fib* (Video 3.8.3)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$
Lo probaremos usando el principio de inducción fuerte:

- $P(n) \equiv fib(n) > n$
- **[Caso base]** Demostrar $P(4) \wedge P(5)$ (¿Por qué?)
  $P(4) \equiv fib(4) > 4 \equiv 5 > 4 \equiv true$
  $P(5) \equiv fib(5) > 5 \equiv 8 > 5 \equiv true$
  Por tanto: $P(4) \wedge P(5) \equiv true$
- **[Caso de inducción]** Demostrar $(\forall j | 4 \leq j \leq k : P(j)) \vdash P(k+1)$ para $k \geq 5$.

Teo $(P(k+1))$:	$fib(k+1) > (k+1)$	
Hip. Ind. $((\forall j \| 4 \leq j \leq k : P(j)))$:	$HI : (\forall j \| 4 \leq j \leq k : fib(j) > j)$	
	Exp.	Just.
	$fib(k+1)$	
$=$	$fib(k-1) + fib(k)$	**Def. de** *fib*
$>$	$(k-1) + k$	**HI**
$>$	$k+1$	$(k-1) > 1$ **pues** $k \geq 5$
		$\diamond$

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 4 : fib(n) > n$

Probando una propiedad sobre *hanoi* (Video 3.8.4)

Probar que $\forall n | n \in \mathbb{N} \land n \geq 1 : hanoi(n) = 2^n - 1$
Lo probaremos usando el principio de inducción:

- $P(n) \equiv hanoi(n) = 2^n - 1$
- **[Caso base]** Demostrar $P(1)$
  $P(1) \equiv hanoi(1) = 2^1 - 1 \equiv 1 = 1 \equiv true$
  Por tanto: $P(1) \equiv true$
- **[Caso de inducción]** Demostrar $P(k) \vdash P(k+1)$ para $k \geq 1$.
- Por tanto, $\forall n | n \in \mathbb{N} \land n \geq 1 : hanoi(n) = 2^n - 1$

Probando una propiedad sobre *hanoi* (Video 3.8.4)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$
Lo probaremos usando el principio de inducción:

- $P(n) \equiv hanoi(n) = 2^n - 1$
- **[Caso base]** Demostrar $P(1)$
  $P(1) \equiv hanoi(1) = 2^1 - 1 \equiv 1 = 1 \equiv true$
  Por tanto: $P(1) \equiv true$
- **[Caso de inducción]** Demostrar $P(k) \vdash P(k+1)$ para $k \geq 1$.

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$

Probando una propiedad sobre *hanoi* (Video 3.8.4)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$
Lo probaremos usando el principio de inducción:

- $P(n) \equiv hanoi(n) = 2^n - 1$
- **[Caso base]** Demostrar $P(1)$
  $P(1) \equiv hanoi(1) = 2^1 - 1 \equiv 1 = 1 \equiv true$
  Por tanto: $P(1) \equiv true$
- **[Caso de inducción]** Demostrar $P(k) \vdash P(k+1)$ para $k \geq 1$.
- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$

Probando una propiedad sobre *hanoi* (Video 3.8.4)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$
Lo probaremos usando el principio de inducción:

- $P(n) \equiv hanoi(n) = 2^n - 1$
- **[Caso base]** Demostrar $P(1)$
  $P(1) \equiv hanoi(1) = 2^1 - 1 \equiv 1 = 1 \equiv true$
  Por tanto: $P(1) \equiv true$
- **[Caso de inducción]** Demostrar $P(k) \vdash P(k+1)$ para $k \geq 1$.
- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$

Probando una propiedad sobre *hanoi* (Video 3.8.4)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$
Lo probaremos usando el principio de inducción:

- $P(n) \equiv hanoi(n) = 2^n - 1$
- **[Caso base]** Demostrar $P(1)$
  $P(1) \equiv hanoi(1) = 2^1 - 1 \equiv 1 = 1 \equiv true$
  Por tanto: $P(1) \equiv true$
- **[Caso de inducción]** Demostrar $P(k) \vdash P(k+1)$ para $k \geq 1$.

Tpo [$P(k+1)$]:	$hanoi(k+1) = 2^{k+1} - 1$	
Hip. Ind. ($P(k)$]:	$HI : hanoi(k) = 2^k - 1$	
	Exp.	Just.
	$hanoi(k+1)$	
$=$	$2hanoi(k) + 1$	Def. de *hanoi*
$=$	$2(2^k - 1) + 1$	HI
$=$	$2^{k+1} - 2 + 1$	Aritmética
$=$	$2^{k+1} - 1$	Aritmética
		✓

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$

Probando una propiedad sobre *hanoi* (Video 3.8.4)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$

Lo probaremos usando el principio de inducción:

- $P(n) \equiv hanoi(n) = 2^n - 1$
- **[Caso base]** Demostrar $P(1)$
  $P(1) \equiv hanoi(1) = 2^1 - 1 \equiv 1 = 1 \equiv true$
  Por tanto: $P(1) \equiv true$
- **[Caso de inducción]** Demostrar $P(k) \vdash P(k+1)$ para $k \geq 1$.

Teo ($P(k+1)$):	$hanoi(k+1) = 2^{k+1} - 1$	
Hip. Ind. ($P(k)$):	$HI : hanoi(k) = 2^k - 1$	
	Exp.	Just.
	$hanoi(k+1)$	
$=$	$2hanoi(k) + 1$	Def. de *hanoi*
$=$	$2(2^k - 1) + 1$	HI
$=$	$2^{k+1} - 2 + 1$	Aritmética
$=$	$2^{k+1} - 1$	Aritmética
		◇

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$

Probando una propiedad sobre *hanoi* (Video 3.8.4)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$
Lo probaremos usando el principio de inducción:

- $P(n) \equiv hanoi(n) = 2^n - 1$
- **[Caso base]** Demostrar $P(1)$
  $P(1) \equiv hanoi(1) = 2^1 - 1 \equiv 1 = 1 \equiv true$
  Por tanto: $P(1) \equiv true$
- **[Caso de inducción]** Demostrar $P(k) \vdash P(k+1)$ para $k \geq 1$.

Teo ($P(k+1)$):	$hanoi(k+1) = 2^{k+1} - 1$	
Hip. Ind. ($P(k)$):	$HI : hanoi(k) = 2^k - 1$	
	Exp.	Just.
	$hanoi(k+1)$	
$=$	$2hanoi(k) + 1$	**Def. de** *hanoi*
$=$	$2(2^k - 1) + 1$	**HI**
$=$	$2^{k+1} - 2 + 1$	**Aritmética**
$=$	$2^{k+1} - 1$	**Aritmética**
		$\diamond$

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$

Probando una propiedad sobre *hanoi* (Video 3.8.4)

Probar que $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$
Lo probaremos usando el principio de inducción:

- $P(n) \equiv hanoi(n) = 2^n - 1$
- **[Caso base]** Demostrar $P(1)$
  $P(1) \equiv hanoi(1) = 2^1 - 1 \equiv 1 = 1 \equiv true$
  Por tanto: $P(1) \equiv true$
- **[Caso de inducción]** Demostrar $P(k) \vdash P(k+1)$ para $k \geq 1$.

Teo ($P(k+1)$):	$hanoi(k+1) = 2^{k+1} - 1$	
Hip. Ind. ($P(k)$):	$HI : hanoi(k) = 2^k - 1$	
	Exp.	Just.
	$hanoi(k+1)$	
$=$	$2hanoi(k) + 1$	**Def. de** *hanoi*
$=$	$2(2^k - 1) + 1$	**HI**
$=$	$2^{k+1} - 2 + 1$	**Aritmética**
$=$	$2^{k+1} - 1$	**Aritmética**
		$\diamond$

- Por tanto, $\forall n | n \in \mathbb{N} \wedge n \geq 1 : hanoi(n) = 2^n - 1$

Probando una propiedad sobre *hanoi* (Video 3.8.4)

Probar que $\forall n | n \in \mathbb{N} \land n \geq 1 : hanoi(n) = 2^n - 1$
Lo probaremos usando el principio de inducción:

- $P(n) \equiv hanoi(n) = 2^n - 1$
- **[Caso base]** Demostrar $P(1)$
  $P(1) \equiv hanoi(1) = 2^1 - 1 \equiv 1 = 1 \equiv true$
  Por tanto: $P(1) \equiv true$
- **[Caso de inducción]** Demostrar $P(k) \vdash P(k+1)$ para $k \geq 1$.

Teo ($P(k+1)$):	$hanoi(k+1) = 2^{k+1} - 1$	
Hip. Ind. ($P(k)$):	$HI : hanoi(k) = 2^k - 1$	
	Exp.	Just.
	$hanoi(k+1)$	
$=$	$2hanoi(k) + 1$	**Def. de** ***hanoi***
$=$	$2(2^k - 1) + 1$	**HI**
$=$	$2^{k+1} - 2 + 1$	**Aritmética**
$=$	$2^{k+1} - 1$	**Aritmética**
		$\diamond$

- Por tanto, $\forall n | n \in \mathbb{N} \land n \geq 1 : hanoi(n) = 2^n - 1$

Plan

1. Motivación
2. Definiciones recursivas
   - . . . de Funciones
   - . . . de Conjuntos
3. Las secuencias y los árboles como estructuras discretas
   - ¿Qué entendemos por *estructuras*?
   - Las secuencias
   - Los árboles binarios
4. Inducción estructural
   - Inducción estructural
   - Teoremas sobre secuencias
   - Teoremas sobre árboles

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \Rightarrow (x+y) \in A$
  - Enumere 5 elementos de A
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A.
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo).
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A:
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
    - $3 \in A$
    - $x \in A \land y \in A \implies (x + y) \in A$
    - Enumere 5 elementos de A: $3, 6, 9, 12, 15, \ldots$
    - ¿Cómo podría describir los elementos de A?
    - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \land x \in A \land y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A: $3, 6, 9, 12, 15, \ldots$
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x+y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, ...
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \rightarrow \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, ...
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, ...
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, . . .
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, ...
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9) = coc(3+6) = coc(3) + coc(6)$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x+y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, ...
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9)$ $= coc(3+6) = coc(3) + coc(6) = coc(3) + coc(3+3)$ $= coc(3) + coc(3) + coc(3) = 1 + 1 + 1 = 3$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, . . .
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9) = coc(3 + 6) = coc(3) + coc(6) = coc(3) + coc(3 + 3)$ $= coc(3) + coc(3) + coc(3) = 1 + 1 + 1 = 3$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x+y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, . . .
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9) = coc(3+6) = coc(3) + coc(6) = coc(3) + coc(3+3)$
$= coc(3) + coc(3) + coc(3) = 1 + 1 + 1 = 3$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, . . .
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \rightarrow \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9) = coc(3 + 6) = coc(3) + coc(6) = coc(3) + coc(3 + 3)$ $= coc(3) + coc(3) + coc(3) = 1 + 1 + 1 = 3$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x+y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, ...
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9) = coc(3+6) = coc(3) + coc(6) = coc(3) + coc(3+3)$ $= coc(3) + coc(3) + coc(3) = 1 + 1 + 1 = 3$

Definición recursiva de conjuntos

- Toda definición recursiva de un conjunto tiene dos tipos de reglas: (1)las básicas, que especifican elementos que pertenecen al conjunto y (1)las recursivas, que especifican cómo construir elementos del conjunto a partir de elementos ya conocidos del conjunto.
- Sea A el conjunto definido así:
  - $3 \in A$
  - $x \in A \wedge y \in A \implies (x + y) \in A$
  - Enumere 5 elementos de A: 3, 6, 9, 12, 15, . . .
  - ¿Cómo podría describir los elementos de A?
  - Todo elemento de A es múltiplo de 3 (pendiente demostrarlo)
- Sobre un conjunto definido recursivamente, es natural definir funciones (u operaciones) recursivamente. Por ejemplo, considere la función $coc : A \to \mathbb{N}$:

$$coc(a) = \begin{cases} 1 & a = 3 \\ coc(x) + coc(y) & a = x + y \wedge x \in A \wedge y \in A \end{cases}$$

- Calcule $coc(9) = coc(3 + 6) = coc(3) + coc(6) = coc(3) + coc(3 + 3)$ $= coc(3) + coc(3) + coc(3) = 1 + 1 + 1 = 3$

¿Entendimos las definiciones recursivas de funciones y conjuntos? (1)

[Socrative]Para cada función $f : \mathbb{N} \to \mathbb{N}$ definida recursivamente, calcule los valores solicitados:

1. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ f(n-1) + 2 & n \geq 1 \end{cases}$$
Calcule $f(1), f(2), f(6)$.
2. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 2^{f(n-1)} & n \geq 1 \end{cases}$$
Calcule $f(1), f(2), f(4)$.
3. Sea
$$f(n) = \begin{cases} 3 & n = 0 \\ 3f(n-1) + 7 & n \geq 1 \end{cases}$$
Calcule $f(1), f(2), f(3)$.

¿Entendimos las definiciones recursivas de funciones y conjuntos? (1)

[Socrative]Para cada función $f : \mathbb{N} \to \mathbb{N}$ definida recursivamente, calcule los valores solicitados:

1. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ f(n-1) + 2 & n \geq 1 \end{cases}$$
Calcule $f(1), f(2), f(6)$.
2. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 2^{f(n-1)} & n \geq 1 \end{cases}$$
Calcule $f(1), f(2), f(4)$.
3. Sea
$$f(n) = \begin{cases} 3 & n = 0 \\ 3f(n-1) + 7 & n \geq 1 \end{cases}$$
Calcule $f(1), f(2), f(3)$.

¿Entendimos las definiciones recursivas de funciones y conjuntos? (1)

[Socrative]Para cada función $f : \mathbb{N} \to \mathbb{N}$ definida recursivamente, calcule los valores solicitados:

1. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ f(n-1) + 2 & n \geq 1 \end{cases}$$
Calcule $f(1), f(2), f(6)$.
2. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 2^{f(n-1)} & n \geq 1 \end{cases}$$
Calcule $f(1), f(2), f(4)$.
3. Sea
$$f(n) = \begin{cases} 3 & n = 0 \\ 3f(n-1) + 7 & n \geq 1 \end{cases}$$
Calcule $f(1), f(2), f(3)$.

¿Entendimos las definiciones recursivas de funciones y conjuntos? (2)

[Socrative]Para cada función $f : \mathbb{N} \to \mathbb{Z}$ definida recursivamente, calcule los valores solicitados:

1. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ f(n-1) - f(n-2) & n \geq 2 \end{cases}$$
Calcule $f(2), f(3), f(5)$.
2. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ f(n-1)f(n-2) & n \geq 2 \end{cases}$$
Calcule $f(2), f(3), f(5)$.
3. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ f(n-1)/f(n-2) & n \geq 2 \end{cases}$$
Calcule $f(2), f(3), f(5)$.

¿Entendimos las definiciones recursivas de funciones y conjuntos? (2)

[Socrative]Para cada función $f : \mathbb{N} \rightarrow \mathbb{Z}$ definida recursivamente, calcule los valores solicitados:

1. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ f(n-1) - f(n-2) & n \geq 2 \end{cases}$$
Calcule $f(2), f(3), f(5)$.
2. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ f(n-1)f(n-2) & n \geq 2 \end{cases}$$
Calcule $f(2), f(3), f(5)$.
3. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ f(n-1)/f(n-2) & n \geq 2 \end{cases}$$
Calcule $f(2), f(3), f(5)$.

¿Entendimos las definiciones recursivas de funciones y conjuntos? (2)

[Socrative]Para cada función $f : \mathbb{N} \to \mathbb{Z}$ definida recursivamente, calcule los valores solicitados:

1. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ f(n-1) - f(n-2) & n \geq 2 \end{cases}$$
Calcule $f(2), f(3), f(5)$.
2. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ f(n-1)f(n-2) & n \geq 2 \end{cases}$$
Calcule $f(2), f(3), f(5)$.
3. Sea
$$f(n) = \begin{cases} 1 & n = 0 \\ 1 & n = 1 \\ f(n-1)/f(n-2) & n \geq 2 \end{cases}$$
Calcule $f(2), f(3), f(5)$.

Plan

Estructuras: Conjuntos + Operaciones (Video 3.8.5)

- Las estructuras de datos usadas en informática se representan formalmente como estructuras algebráicas, es decir, Conjuntos y Operaciones sobre ellos.
- Muchas de estas estructuras se representan con conjuntos definidos recursivamente y operaciones (funciones) sobre esos conjuntos que también son naturalmente recursivas.
- Estas estructuras se usan para representar de manera genérica (independiente de la implementación) estructuras de datos que se usan en informática: Secuencias y Árboles, por ejemplo. Y en general, para describir los Tipos Abstractos de Datos (TAD).
- Entonces, se pueden estudiar estas estructuras y sus propiedades de forma abstracta. Cualquier implementación deberá cumplir esas propiedades.
- Como ejemplo, vamos a definir las estructuras denominadas Secuencias y Árboles Binarios

Estructuras: Conjuntos + Operaciones (Video 3.8.5)

- Las estructuras de datos usadas en informática se representan formalmente como estructuras algebráicas, es decir, Conjuntos y Operaciones sobre ellos.
- Muchas de estas estructuras se representan con conjuntos definidos recursivamente y operaciones (funciones) sobre esos conjuntos que también son naturalmente recursivas.
- Estas estructuras se usan para representar de manera genérica (independiente de la implementación) estructuras de datos que se usan en informática: Secuencias y Árboles, por ejemplo. Y en general, para describir los Tipos Abstractos de Datos (TAD).
- Entonces, se pueden estudiar estas estructuras y sus propiedades de forma abstracta. Cualquier implementación deberá cumplir esas propiedades.
- Como ejemplo, vamos a definir las estructuras denominadas Secuencias y Árboles Binarios

Estructuras: Conjuntos + Operaciones (Video 3.8.5)

- Las estructuras de datos usadas en informática se representan formalmente como estructuras algebráicas, es decir, Conjuntos y Operaciones sobre ellos.
- Muchas de estas estructuras se representan con conjuntos definidos recursivamente y operaciones (funciones) sobre esos conjuntos que también son naturalmente recursivas.
- Estas estructuras se usan para representar de manera genérica (independiente de la implementación) estructuras de datos que se usan en informática: Secuencias y Árboles, por ejemplo. Y en general, para describir los Tipos Abstractos de Datos (TAD).
- Entonces, se pueden estudiar estas estructuras y sus propiedades de forma abstracta. Cualquier implementación deberá cumplir esas propiedades.
- Como ejemplo, vamos a definir las estructuras denominadas Secuencias y Árboles Binarios

Estructuras: Conjuntos + Operaciones (Video 3.8.5)

- Las estructuras de datos usadas en informática se representan formalmente como estructuras algebráicas, es decir, Conjuntos y Operaciones sobre ellos.
- Muchas de estas estructuras se representan con conjuntos definidos recursivamente y operaciones (funciones) sobre esos conjuntos que también son naturalmente recursivas.
- Estas estructuras se usan para representar de manera genérica (independiente de la implementación) estructuras de datos que se usan en informática: Secuencias y Árboles, por ejemplo. Y en general, para describir los Tipos Abstractos de Datos (TAD).
- Entonces, se pueden estudiar estas estructuras y sus propiedades de forma abstracta. Cualquier implementación deberá cumplir esas propiedades.
- Como ejemplo, vamos a definir las estructuras denominadas Secuencias y Árboles Binarios

Estructuras: Conjuntos + Operaciones (Video 3.8.5)

- Las estructuras de datos usadas en informática se representan formalmente como estructuras algebráicas, es decir, Conjuntos y Operaciones sobre ellos.
- Muchas de estas estructuras se representan con conjuntos definidos recursivamente y operaciones (funciones) sobre esos conjuntos que también son naturalmente recursivas.
- Estas estructuras se usan para representar de manera genérica (independiente de la implementación) estructuras de datos que se usan en informática: Secuencias y Árboles, por ejemplo. Y en general, para describir los Tipos Abstractos de Datos (TAD).
- Entonces, se pueden estudiar estas estructuras y sus propiedades de forma abstracta. Cualquier implementación deberá cumplir esas propiedades.
- Como ejemplo, vamos a definir las estructuras denominadas Secuencias y Árboles Binarios

Plan

Motivación - Definiciones (Video 3.8.6)

- Los conjuntos se usan para modelar datos de ciertos tipos (enteros, reales, caracteres, ...), normalmente de tamaño fijo. Para modelar datos de tamaño arbitrariamente grandes, se usa una estructura discreta denominada secuencia.
- Con las secuencias se desea representar listas ordenadas de elementos de algún conjunto. Una secuencia permite "empaquetar" en un sólo "dato" múltiples elementos de un conjunto de forma ordenada.
- Formalmente, una secuencia s de elementos de un conjunto A es una función

$$s : \{i \in \mathbb{N} : 0 \leq i < n\} \rightarrow A$$

$s(0)$ es el primer elemento de la secuencia, $s(1)$ es el segundo, y así sucesivamente hasta $s(n-1)$ que es el último elemento de la secuencia. Se suele escribir

$$< s_0, s_1, \ldots, s_{n-1} >$$

Por definición toda secuencia tiene un número finito de elementos

- Cuando se desea modelar una lista ordenada de un número infinito de elementos se utiliza una sucesión, que no es más que

$$s : \mathbb{N} \rightarrow A$$

Motivación - Definiciones (Video 3.8.6)

- Los conjuntos se usan para modelar datos de ciertos tipos (enteros, reales, caracteres, ...), normalmente de tamaño fijo. Para modelar datos de tamaño arbitrariamente grandes, se usa una estructura discreta denominada secuencia.
- Con las secuencias se desea representar listas ordenadas de elementos de algún conjunto. Una secuencia permite "empaquetar" en un sólo "dato" múltiples elementos de un conjunto de forma ordenada.
- Formalmente, una secuencia s de elementos de un conjunto A es una función

$$s : \{i \in \mathbb{N} : 0 \leq i < n\} \rightarrow A$$

$s(0)$ es el primer elemento de la secuencia, $s(1)$ es el segundo, y así sucesivamente hasta $s(n-1)$ que es el último elemento de la secuencia. Se suele escribir

$$< s_0, s_1, \ldots, s_{n-1} >$$

Por definición toda secuencia tiene un número finito de elementos
- Cuando se desea modelar una lista ordenada de un número infinito de elementos se utiliza una sucesión, que no es más que

$$s : \mathbb{N} \rightarrow A$$

Motivación - Definiciones (Video 3.8.6)

- Los conjuntos se usan para modelar datos de ciertos tipos (enteros, reales, caracteres, ...), normalmente de tamaño fijo. Para modelar datos de tamaño arbitrariamente grandes, se usa una estructura discreta denominada secuencia.
- Con las secuencias se desea representar listas ordenadas de elementos de algún conjunto. Una secuencia permite "empaquetar" en un sólo "dato" múltiples elementos de un conjunto de forma ordenada.
- Formalmente, una secuencia s de elementos de un conjunto A es una función

  $$s : \{i \in \mathbb{N} : 0 \leq i < n\} \rightarrow A$$

  $s(0)$ es el primer elemento de la secuencia, $s(1)$ es el segundo, y así sucesivamente hasta $s(n-1)$ que es el último elemento de la secuencia. Se suele escribir

  $$< s_0, s_1, \ldots, s_{n-1} >$$

  Por definición toda secuencia tiene un número finito de elementos
- Cuando se desea modelar una lista ordenada de un número infinito de elementos se utiliza una sucesión, que no es más que

  $$s : \mathbb{N} \rightarrow A$$

Motivación - Definiciones (Video 3.8.6)

- Los conjuntos se usan para modelar datos de ciertos tipos (enteros, reales, caracteres, ...), normalmente de tamaño fijo. Para modelar datos de tamaño arbitrariamente grandes, se usa una estructura discreta denominada secuencia.
- Con las secuencias se desea representar listas ordenadas de elementos de algún conjunto. Una secuencia permite "empaquetar" en un sólo "dato" múltiples elementos de un conjunto de forma ordenada.
- Formalmente, una secuencia s de elementos de un conjunto A es una función

$$s : \{i \in \mathbb{N} : 0 \leq i < n\} \rightarrow A$$

$s(0)$ es el primer elemento de la secuencia, $s(1)$ es el segundo, y así sucesivamente hasta $s(n-1)$ que es el último elemento de la secuencia.
Se suele escribir

$$< s_0, s_1, \ldots, s_{n-1} >$$

Por definición toda secuencia tiene un número finito de elementos
- Cuando se desea modelar una lista ordenada de un número infinito de elementos se utiliza una sucesión, que no es más que

$$s : \mathbb{N} \rightarrow A$$

Definición recursiva de secuencia (Video 3.8.6)

El conjunto de secuencias de elementos del conjunto A ($Seq[A]$) también se puede definir recursivamente, así:

- ϵ representa la secuencia sin elementos o secuencia vacía
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$ representa la secuencia cuyo primer elemento es a y los otros están en el orden en que estaban en s, es decir,

$$t_0 = a, t_1 = s_0, \ldots, t_n = s_{n-1}$$

  Visto de otra manera

$$a \triangleleft < a_0, a_1, \ldots, a_{n-1} > = < a, a_0, a_1, \ldots, a_{n-1} >$$

- Nótese que la secuencia $< a_0, a_1, \ldots, a_{n-1} >$ se puede escribir como

$$a_0 \triangleleft (a_1 \triangleleft (a_2 \triangleleft \ldots \triangleleft (a_{n-1} \triangleleft \epsilon) \ldots)$$

  y se escribe por convención

$$a_0 \triangleleft a_1 \triangleleft a_2 \triangleleft \ldots \triangleleft a_{n-1} \triangleleft \epsilon$$

- Tenemos ahora un mecanismo para construir secuencias incrementalmente.

Definición recursiva de secuencia (Video 3.8.6)

El conjunto de secuencias de elementos del conjunto A ($Seq[A]$) también se puede definir recursivamente, así:

- ϵ representa la secuencia sin elementos o secuencia vacía
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$ representa la secuencia cuyo primer elemento es a y los otros están en el orden en que estaban en s, es decir,

$$t_0 = a, t_1 = s_0, \ldots, t_n = s_{n-1}$$

Visto de otra manera

$$a \triangleleft < a_0, a_1, \ldots, a_{n-1} > = < a, a_0, a_1, \ldots, a_{n-1} >$$

- Nótese que la secuencia $< a_0, a_1, \ldots, a_{n-1} >$ se puede escribir como

$$a_0 \triangleleft (a_1 \triangleleft (a_2 \triangleleft \ldots \triangleleft (a_{n-1} \triangleleft \epsilon) \ldots)$$

y se escribe por convención

$$a_0 \triangleleft a_1 \triangleleft a_2 \triangleleft \ldots \triangleleft a_{n-1} \triangleleft \epsilon$$

- Tenemos ahora un mecanismo para construir secuencias incrementalmente.

Definición recursiva de secuencia (Video 3.8.6)

El conjunto de secuencias de elementos del conjunto A ($Seq[A]$) también se puede definir recursivamente, así:

- ϵ representa la secuencia sin elementos o secuencia vacía
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \lhd s \in Seq[A]$ representa la secuencia cuyo primer elemento es a y los otros están en el orden en que estaban en s, es decir,

  $$t_0 = a, t_1 = s_0, \ldots, t_n = s_{n-1}$$

  Visto de otra manera

  $$a \lhd < a_0, a_1, \ldots, a_{n-1} >=< a, a_0, a_1, \ldots, a_{n-1} >$$

- Nótese que la secuencia $< a_0, a_1, \ldots, a_{n-1} >$ se puede escribir como

  $$a_0 \lhd (a_1 \lhd (a_2 \lhd \ldots \lhd (a_{n-1} \lhd \epsilon) \ldots)$$

  y se escribe por convención

  $$a_0 \lhd a_1 \lhd a_2 \lhd \ldots \lhd a_{n-1} \lhd \epsilon$$

- Tenemos ahora un mecanismo para construir secuencias incrementalmente.

Definición recursiva de secuencia (Video 3.8.6)

El conjunto de secuencias de elementos del conjunto A ($Seq[A]$) también se puede definir recursivamente, así:

- ϵ representa la secuencia sin elementos o secuencia vacía
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$ representa la secuencia cuyo primer elemento es a y los otros están en el orden en que estaban en s, es decir,

  $$t_0 = a, t_1 = s_0, \ldots, t_n = s_{n-1}$$

  Visto de otra manera

  $$a \triangleleft < a_0, a_1, \ldots, a_{n-1} > = < a, a_0, a_1, \ldots, a_{n-1} >$$

- Nótese que la secuencia $< a_0, a_1, \ldots, a_{n-1} >$ se puede escribir como

  $$a_0 \triangleleft (a_1 \triangleleft (a_2 \triangleleft \ldots \triangleleft (a_{n-1} \triangleleft \epsilon) \ldots)$$

  y se escribe por convención

  $$a_0 \triangleleft a_1 \triangleleft a_2 \triangleleft \ldots \triangleleft a_{n-1} \triangleleft \epsilon$$

- Tenemos ahora un mecanismo para construir secuencias incrementalmente.

Definición recursiva de secuencia (Video 3.8.6)

El conjunto de secuencias de elementos del conjunto A ($Seq[A]$) también se puede definir recursivamente, así:

- ϵ representa la secuencia sin elementos o secuencia vacía
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$ representa la secuencia cuyo primer elemento es a y los otros están en el orden en que estaban en s, es decir,

$$t_0 = a, t_1 = s_0, \ldots, t_n = s_{n-1}$$

  Visto de otra manera

$$a \triangleleft < a_0, a_1, \ldots, a_{n-1} > = < a, a_0, a_1, \ldots, a_{n-1} >$$

- Nótese que la secuencia $< a_0, a_1, \ldots, a_{n-1} >$ se puede escribir como

$$a_0 \triangleleft (a_1 \triangleleft (a_2 \triangleleft \ldots \triangleleft (a_{n-1} \triangleleft \epsilon) \ldots)$$

  y se escribe por convención

$$a_0 \triangleleft a_1 \triangleleft a_2 \triangleleft \ldots \triangleleft a_{n-1} \triangleleft \epsilon$$

- Tenemos ahora un mecanismo para construir secuencias incrementalmente.

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \lhd s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función): f	Caso f(ϵ)	Caso $f(x \lhd s)$	Tipo

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \lhd s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función): f	Caso f(ϵ)	Caso $f(x \lhd s)$	Tipo

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \lhd s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \lhd s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \lhd dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \lhd s) \oplus t = x \lhd (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \lhd \epsilon$	Total

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

Operaciones sobre secuencias

- Sea $s =< 4,2,5 >\in Seq[\mathbb{N}]$. $7 \lhd s =< 7,4,2,5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \lhd s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \lhd dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \lhd s) \oplus t = x \lhd (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \lhd \epsilon$	Total

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \lhd s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \lhd s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \lhd dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \lhd s) \oplus t = x \lhd (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \lhd \epsilon$	Total

- Calcula

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \lhd s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \lhd s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \lhd dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \lhd s) \oplus t = x \lhd (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \lhd \epsilon$	Total

- Calcule:

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \lhd s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \lhd s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \lhd dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \lhd s) \oplus t = x \lhd (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \lhd \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $size(s) = 3$
$last(s) = 5$ $dlast(s) = < 4, 2 >$
$rev(s) = < 5, 2, 4 >$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \rightarrow A$		x	Parcial
$tail : Seq[A] \rightarrow Seq[A]$		s	Parcial
$size : Seq[A] \rightarrow \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \rightarrow A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \rightarrow Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \rightarrow Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \rightarrow Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:
  $tail(s) =< 2, 5 >$ $size(s) = 3$
  $last(s) = 5$ $dlast(s) =< 4, 2 >$
  $rev(s) =< 5, 2, 4 >$ $(s \oplus < 3, 1 >) =< 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s = < 4, 2, 5 > \in Seq[\mathbb{N}]$. $7 \triangleleft s = < 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:
  $tail(s) = < 2, 5 >$ $\quad$ $size(s) = 3$
  $last(s) = 5$ $\quad$ $dlast(s) = < 4, 2 >$
  $rev(s) = < 5, 2, 4 >$ $\quad$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $size(s) = 3$

$last(s) = 5$ $dlast(s) = < 4, 2 >$

$rev(s) = < 5, 2, 4 >$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $size(s) = 3$

$last(s) = 5$ $dlast(s) = < 4, 2 >$

$rev(s) = < 5, 2, 4 >$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función): f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $\quad$ $size(s) = 3$

$last(s) = 5$ $\quad$ $dlast(s) = < 4, 2 >$

$rev(s) = < 5, 2, 4 >$ $\quad$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $\quad$ $size(s) = 3$

$last(s) = 5$ $\quad$ $dlast(s) = < 4, 2 >$

$rev(s) = < 5, 2, 4 >$ $\quad$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4,2,5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7,4,2,5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \rightarrow A$		x	Parcial
$tail : Seq[A] \rightarrow Seq[A]$		s	Parcial
$size : Seq[A] \rightarrow \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \rightarrow A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \rightarrow Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \rightarrow Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \rightarrow Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:

$tail(s) = < 2,5 >$ $\quad$ $size(s) = 3$

$last(s) = 5$ $\quad$ $dlast(s) = < 4,2 >$

$rev(s) = < 5,2,4 >$ $\quad$ $(s \oplus < 3,1 >) = < 4,2,5,3,1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \lhd s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función): f	Caso f(ϵ)	Caso $f(x \lhd s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \lhd dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \lhd s) \oplus t = x \lhd (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \lhd \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $\quad$ $size(s) = 3$

$last(s) = 5$ $\quad$ $dlast(s) = < 4, 2 >$

$rev(s) = < 5, 2, 4 >$ $\quad$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \lhd s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \lhd s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \lhd dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \lhd s) \oplus t = x \lhd (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \lhd \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $\quad$ $size(s) = 3$

$last(s) = 5$ $\quad$ $dlast(s) = < 4, 2 >$

$rev(s) = < 5, 2, 4 >$ $\quad$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $\quad$ $size(s) = 3$

$last(s) = 5$ $\quad$ $dlast(s) = < 4, 2 >$

$rev(s) = < 5, 2, 4 >$ $\quad$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $\quad$ $size(s) = 3$

$last(s) = 5$ $\quad$ $dlast(s) = < 4, 2 >$

$rev(s) = < 5, 2, 4 >$ $\quad$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Operaciones sobre secuencias

- Sea $s =< 4, 2, 5 >\in Seq[\mathbb{N}]$. $7 \triangleleft s =< 7, 4, 2, 5 >$
- Hay unas operaciones (funciones) muy útiles sobre secuencias, que se definen recursivamente por casos, sobre la forma como se construyen secuencias:

Operación(función):f	Caso f(ϵ)	Caso $f(x \triangleleft s)$	Tipo
$head : Seq[A] \to A$		x	Parcial
$tail : Seq[A] \to Seq[A]$		s	Parcial
$size : Seq[A] \to \mathbb{N}$	0	$1 + size(s)$	Total
$last : Seq[A] \to A$		$\begin{cases} x & \text{Si } s = \epsilon \\ last(s) & \text{sino} \end{cases}$	Parcial
$dlast : Seq[A] \to Seq[A]$		$\begin{cases} \epsilon & \text{Si } s = \epsilon \\ x \triangleleft dlast(s) & \text{sino} \end{cases}$	Parcial
$\oplus : Seq[A] \times Seq[A] \to Seq[A]$	$\epsilon \oplus t = t$	$(x \triangleleft s) \oplus t = x \triangleleft (s \oplus t)$	Total
$rev : Seq[A] \to Seq[A]$	ϵ	$rev(s) \oplus x \triangleleft \epsilon$	Total

- Calcule:

$tail(s) = < 2, 5 >$ $\quad$ $size(s) = 3$

$last(s) = 5$ $\quad$ $dlast(s) = < 4, 2 >$

$rev(s) = < 5, 2, 4 >$ $\quad$ $(s \oplus < 3, 1 >) = < 4, 2, 5, 3, 1 >$

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \wedge t = y \triangleleft t_1 \wedge x = y \wedge s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
- Queda pendiente demostrarlos!

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \wedge t = y \triangleleft t_1 \wedge x = y \wedge s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
- Queda pendiente demostrarlos!

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \wedge t = y \triangleleft t_1 \wedge x = y \wedge s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
  - $size(s \oplus t) = size(s) + size(t)$
  - $s \oplus \epsilon = s$
  - $rev(rev(s)) = s$
  - $rev(s \oplus t) = rev(t) \oplus rev(s)$
  - Se dice que una secuencia s es palíndromo si se lee igual de izquierda a derecha que de derecha a izquierda, es decir:

    $$pal(s) = (s = rev(s))$$

    Se puede demostrar que

    $$pal(s \oplus rev(s)) = true$$
- Queda pendiente demostrarlos!

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \wedge t = y \triangleleft t_1 \wedge x = y \wedge s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
  - $size(s \oplus t) = size(s) + size(t)$
  - $s \oplus \epsilon = s$
  - $rev(rev(s)) = s$
  - $rev(s \oplus t) = rev(t) \oplus rev(s)$
  - Se dice que una secuencia s es palíndrome si se lee igual de izquierda a derecha que de derecha a izquierda, es decir:

    $$pal(s) \equiv (s = rev(s))$$

    Se puede demostrar que:

    $$pal(s \oplus rev(s)) \equiv true$$
- Queda pendiente demostrarlos!

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \wedge t = y \triangleleft t_1 \wedge x = y \wedge s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
  - $size(s \oplus t) = size(s) + size(t)$
  - $s \oplus \epsilon = s$
  - $rev(rev(s)) = s$
  - $rev(s \oplus t) = rev(t) \oplus rev(s)$
  - Se dice que una secuencia s es palíndrome si se lee igual de izquierda a derecha que de derecha a izquierda, es decir:

    $$pal(s) \equiv (s = rev(s))$$

    Se puede demostrar que:

    $$pal(s \oplus rev(s)) \equiv true$$

- Queda pendiente demostrarlos!

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \wedge t = y \triangleleft t_1 \wedge x = y \wedge s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
  - $size(s \oplus t) = size(s) + size(t)$
  - $s \oplus \epsilon = s$
  - $rev(rev(s)) = s$
  - $rev(s \oplus t) = rev(t) \oplus rev(s)$
  - Se dice que una secuencia s es palíndrome si se lee igual de izquierda a derecha que de derecha a izquierda, es decir:

    $$pal(s) \equiv (s = rev(s))$$

    Se puede demostrar que:

    $$pal(s \oplus rev(s)) \equiv true$$

- Queda pendiente demostrarlos!

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \wedge t = y \triangleleft t_1 \wedge x = y \wedge s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
  - $size(s \oplus t) = size(s) + size(t)$
  - $s \oplus \epsilon = s$
  - $rev(rev(s)) = s$
  - $rev(s \oplus t) = rev(t) \oplus rev(s)$
  - Se dice que una secuencia s es palíndrome si se lee igual de izquierda a derecha que de derecha a izquierda, es decir:

    $$pal(s) \equiv (s = rev(s))$$

    Se puede demostrar que:

    $$pal(s \oplus rev(s)) \equiv true$$

- Queda pendiente demostrarlos!

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \wedge t = y \triangleleft t_1 \wedge x = y \wedge s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
  - $size(s \oplus t) = size(s) + size(t)$
  - $s \oplus \epsilon = s$
  - $rev(rev(s)) = s$
  - $rev(s \oplus t) = rev(t) \oplus rev(s)$
  - Se dice que una secuencia s es palíndrome si se lee igual de izquierda a derecha que de derecha a izquierda, es decir:

$$pal(s) \equiv (s = rev(s))$$

    Se puede demostrar que:

$$pal(s \oplus rev(s)) \equiv true$$

- Queda pendiente demostrarlos!

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \land t = y \triangleleft t_1 \land x = y \land s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
  - $size(s \oplus t) = size(s) + size(t)$
  - $s \oplus \epsilon = s$
  - $rev(rev(s)) = s$
  - $rev(s \oplus t) = rev(t) \oplus rev(s)$
  - Se dice que una secuencia s es palíndrome si se lee igual de izquierda a derecha que de derecha a izquierda, es decir:

    $$pal(s) \equiv (s = rev(s))$$

    Se puede demostrar que:

    $$pal(s \oplus rev(s)) \equiv true$$

- Queda pendiente demostrarlos!

Propiedades

- Dados $s, t \in Seq[A]$, se dice que $s = t$ si se da uno de los dos casos siguientes:
  - $s = t = \epsilon$
  - $s = x \triangleleft s_1 \wedge t = y \triangleleft t_1 \wedge x = y \wedge s_1 = t_1$
- Los siguientes son teoremas sobre las secuencias:
  - $size(s \oplus t) = size(s) + size(t)$
  - $s \oplus \epsilon = s$
  - $rev(rev(s)) = s$
  - $rev(s \oplus t) = rev(t) \oplus rev(s)$
  - Se dice que una secuencia s es palíndrome si se lee igual de izquierda a derecha que de derecha a izquierda, es decir:

    $$pal(s) \equiv (s = rev(s))$$

    Se puede demostrar que:

    $$pal(s \oplus rev(s)) \equiv true$$

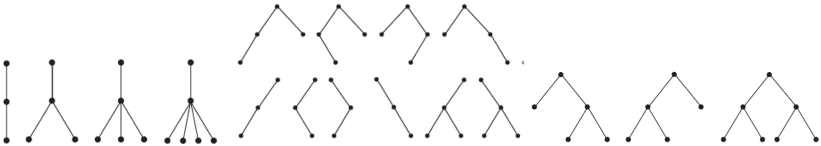
- Queda pendiente demostrarlos!

Plan

1. Motivación
2. Definiciones recursivas
   - . . . de Funciones
   - . . . de Conjuntos
3. Las secuencias y los árboles como estructuras discretas
   - ¿Qué entendemos por *estructuras*?
   - Las secuencias
   - Los árboles binarios
4. Inducción estructural
   - Inducción estructural
   - Teoremas sobre secuencias
   - Teoremas sobre árboles

Motivación - Definiciones

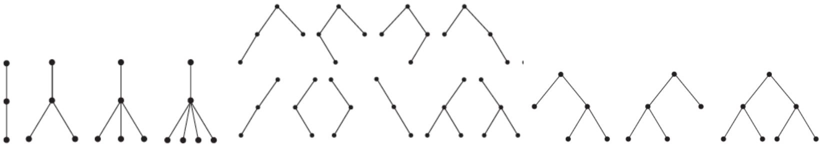
- Con las secuencias se modelaron listas ordenadas de elementos de algún conjunto. Estas estructuras se llaman lineales.
- Hay otras estructuras, un poco más complejas, denominadas ramificadas, que sirven también para modelar datos de tamaño arbitrariamente grandes pero más adecuada para ciertas operaciones y tareas. Estas estructuras son los árboles.
- Hay muchos tipos de árboles: generales, binarios, binarios completos, y muchos más.



- Vamos a definir formalmente la estructura árbol binario completo

Motivación - Definiciones

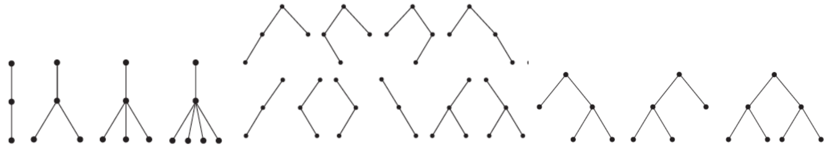
- Con las secuencias se modelaron listas ordenadas de elementos de algún conjunto. Estas estructuras se llaman lineales.
- Hay otras estructuras, un poco más complejas, denominadas ramificadas, que sirven también para modelar datos de tamaño arbitrariamente grandes pero más adecuada para ciertas operaciones y tareas. Estas estructuras son los árboles.
- Hay muchos tipos de árboles: generales, binarios, binarios completos, y muchos más.



- Vamos a definir formalmente la estructura árbol binario completo

Motivación - Definiciones

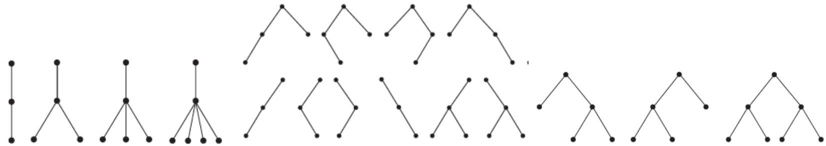
- Con las secuencias se modelaron listas ordenadas de elementos de algún conjunto. Estas estructuras se llaman lineales.
- Hay otras estructuras, un poco más complejas, denominadas ramificadas, que sirven también para modelar datos de tamaño arbitrariamente grandes pero más adecuada para ciertas operaciones y tareas. Estas estructuras son los árboles.
- Hay muchos tipos de árboles: generales, binarios, binarios completos, y muchos más.



- Vamos a definir formalmente la estructura árbol binario completo

Motivación - Definiciones

- Con las secuencias se modelaron listas ordenadas de elementos de algún conjunto. Estas estructuras se llaman lineales.
- Hay otras estructuras, un poco más complejas, denominadas ramificadas, que sirven también para modelar datos de tamaño arbitrariamente grandes pero más adecuada para ciertas operaciones y tareas. Estas estructuras son los árboles.
- Hay muchos tipos de árboles: generales, binarios, binarios completos, y muchos más.



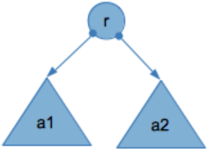
- Vamos a definir formalmente la estructura árbol binario completo

Definición recursiva de árbol binario completo

El conjunto de árboles binarios completos de elementos del conjunto A ($ArBinC[A]$) se define recursivamente, así:

- Dado $r \in A$, $\nabla(r)$ representa el árbol con raíz r y sin hijos (hoja).
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$ representa el árbol binario completo con raíz r, subárbol izquierdo a_1 y subárbol derecho a_2.
  Visto gráficamente:

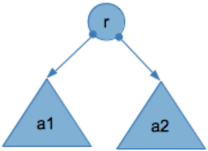
$r \rightarrowtail (a_1, a_2) \in ArBinC[A] =$



- Tenemos ahora un mecanismo para construir árboles binarios completos incrementalmente.

Definición recursiva de árbol binario completo

El conjunto de árboles binarios completos de elementos del conjunto A ($ArBinC[A]$) se define recursivamente, así:

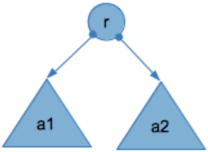
- Dado $r \in A$, $\nabla(r)$ representa el árbol con raíz r y sin hijos (hoja).
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$ representa el árbol binario completo con raíz r, subárbol izquierdo a_1 y subárbol derecho a_2.
  Visto gráficamente:
  $r \rightarrowtail (a_1, a_2) \in ArBinC[A] =$ 

- Tenemos ahora un mecanismo para construir árboles binarios completos incrementalmente.

Definición recursiva de árbol binario completo

El conjunto de árboles binarios completos de elementos del conjunto A ($ArBinC[A]$) se define recursivamente, así:

- Dado $r \in A$, $\nabla(r)$ representa el árbol con raíz r y sin hijos (hoja).
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$ representa el árbol binario completo con raíz r, subárbol izquierdo a_1 y subárbol derecho a_2.
  Visto gráficamente:

  $r \rightarrowtail (a_1, a_2) \in ArBinC[A] =$ 

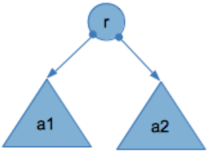
- Tenemos ahora un mecanismo para construir árboles binarios completos incrementalmente.

Definición recursiva de árbol binario completo

El conjunto de árboles binarios completos de elementos del conjunto A ($ArBinC[A]$) se define recursivamente, así:

- Dado $r \in A$, $\nabla(r)$ representa el árbol con raíz r y sin hijos (hoja).
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$ representa el árbol binario completo con raíz r, subárbol izquierdo a_1 y subárbol derecho a_2.
  Visto gráficamente:

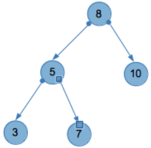
$r \rightarrowtail (a_1, a_2) \in ArBinC[A] =$



- Tenemos ahora un mecanismo para construir árboles binarios completos incrementalmente.

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



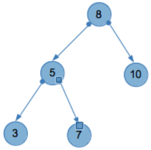
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f$(\nabla(r))$	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \to A$	r	r
$izq : ArBinC[A] \to ArBinC[A]$		a_1
$der : ArBinC[A] \to ArBinC[A]$		a_2
$nodos : ArBinC[A] \to \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \to \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule:

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

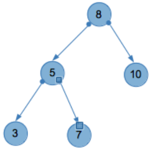


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \to A$	r	r
$izq : ArBinC[A] \to ArBinC[A]$		a_1
$der : ArBinC[A] \to ArBinC[A]$		a_2
$nodos : ArBinC[A] \to \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \to \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

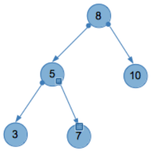


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función): f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

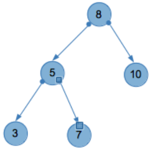


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in$ $ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

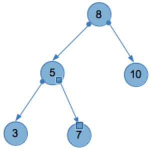


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

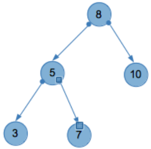


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso $f(\nabla(r))$	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in$ $ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

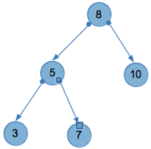


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función): f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

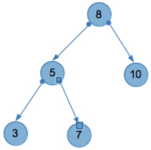


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



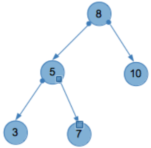
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2

$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

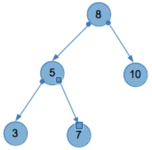


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función): f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \to A$	r	r
$izq : ArBinC[A] \to ArBinC[A]$		a_1
$der : ArBinC[A] \to ArBinC[A]$		a_2
$nodos : ArBinC[A] \to \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \to \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in$ $ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

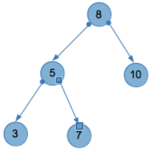


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in$ $ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:

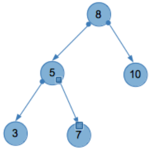


- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \to A$	r	r
$izq : ArBinC[A] \to ArBinC[A]$		a_1
$der : ArBinC[A] \to ArBinC[A]$		a_2
$nodos : ArBinC[A] \to \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \to \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



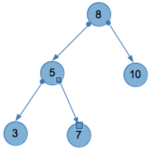
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule:

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in$ $ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



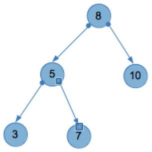
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule:

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



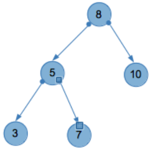
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



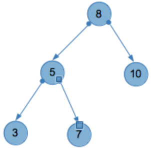
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $der(a) = \nabla(10)$ $nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$ $alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



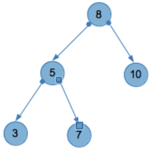
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función): f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \to A$	r	r
$izq : ArBinC[A] \to ArBinC[A]$		a_1
$der : ArBinC[A] \to ArBinC[A]$		a_2
$nodos : ArBinC[A] \to \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \to \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $der(a) = \nabla(10)$
$nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
$alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



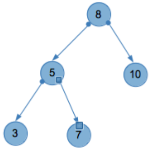
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $\quad izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $\quad der(a) = \nabla(10)$
  $nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
  $alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



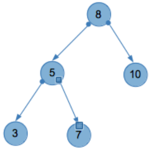
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \to A$	r	r
$izq : ArBinC[A] \to ArBinC[A]$		a_1
$der : ArBinC[A] \to ArBinC[A]$		a_2
$nodos : ArBinC[A] \to \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \to \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $\quad$ $izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $\quad$ $der(a) = \nabla(10)$
$nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
$alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



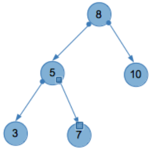
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \to A$	r	r
$izq : ArBinC[A] \to ArBinC[A]$		a_1
$der : ArBinC[A] \to ArBinC[A]$		a_2
$nodos : ArBinC[A] \to \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \to \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $der(a) = \nabla(10)$
$nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
$alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



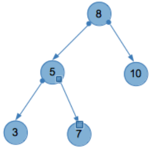
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $der(a) = \nabla(10)$
$nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
$alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



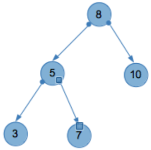
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $\quad izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $\quad der(a) = \nabla(10)$
$nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
$alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



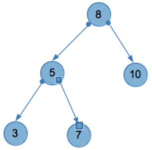
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \to A$	r	r
$izq : ArBinC[A] \to ArBinC[A]$		a_1
$der : ArBinC[A] \to ArBinC[A]$		a_2
$nodos : ArBinC[A] \to \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \to \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $der(a) = \nabla(10)$
  $nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
  $alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in$ $ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



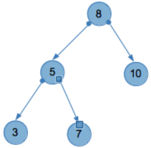
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función): f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $der(a) = \nabla(10)$
  $nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
  $alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in$ $ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



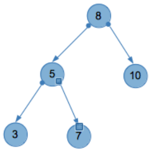
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $der(a) = \nabla(10)$
$nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
$alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



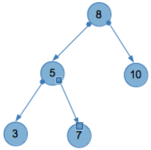
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $der(a) = \nabla(10)$
$nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
$alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



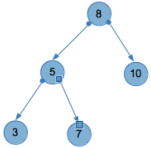
- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \to A$	r	r
$izq : ArBinC[A] \to ArBinC[A]$		a_1
$der : ArBinC[A] \to ArBinC[A]$		a_2
$nodos : ArBinC[A] \to \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \to \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $\quad izq(a) = 5 \rightarrowtail (\nabla(3), \nabla(7))$ $\quad der(a) = \nabla(10)$
$nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
$alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Operaciones sobre árboles binarios completos

Sea $a = 8 \rightarrowtail (5 \rightarrowtail (\nabla(3), \nabla(7)), \nabla(10)) \in$ $ArBinC[\mathbb{N}]$.
a corresponde al árbol binario completo:



- Hay unas operaciones (funciones) muy útiles sobre árboles binarios completos, que se definen recursivamente por casos, sobre la forma como se construyen árboles binarios completos:

Operación(función):f	Caso f($\nabla(r)$)	Caso $f(r \rightarrowtail (a_1, a_2))$
$raiz : ArBinC[A] \rightarrow A$	r	r
$izq : ArBinC[A] \rightarrow ArBinC[A]$		a_1
$der : ArBinC[A] \rightarrow ArBinC[A]$		a_2
$nodos : ArBinC[A] \rightarrow \mathbb{N}$	1	$1 + nodos(a_1) + nodos(a_2)$
$alt : ArBinC[A] \rightarrow \mathbb{N}$	0	$1 + max(alt(a_1), alt(a_2))$

- Calcule: $raiz(a) = 8$ $izq(a) =5 \rightarrowtail (\nabla(3), \nabla(7))$ $der(a) =\nabla(10)$
$nodos(a) = 1 + nodos(5 \rightarrowtail (\nabla(3), \nabla(7))) + nodos(\nabla(10)) = 1 + 3 + 1 = 5$
$alt(a) = 1 + max(alt(5 \rightarrowtail (\nabla(3), \nabla(7))), alt(\nabla(10))) = 1 + max(1, 0) = 2$

Propiedades

- Dados $a \in ArBinC[A]$, se tiene que

$$nodos(a) \leq 2^{alt(a)+1} - 1$$

- Queda pendiente demostrarlo!

Propiedades

- Dados $a \in ArBinC[A]$, se tiene que

$$nodos(a) \leq 2^{alt(a)+1} - 1$$

- Queda pendiente demostrarlo!

Plan

El principio de inducción estructural (Video 3.8.7)

Sea A un conjunto definido recursivamente.
En general, supongamos que queremos probar

$$\forall a | a \in A : P(a)$$

donde $P(a)$ es un predicado sobre A.
Usaremos el principio de inducción estructural de la siguiente manera:

- Establecer clara y formalmente $P(a)$
- **[Caso base]** Demostrar $P(a)$, para todo $a \in A$ definido por una regla básica, usando las técnicas de demostración conocidas.
- **[Caso de inducción]** Demostrar $P(b)$ para todo $b \in A$ construido con una regla recursiva, suponiendo que todos los elementos $a \in A$ usados en la construcción de b cumplen P. Formalmente podemos escribirlo así: $(\forall a \preceq b : P(a)) \vdash P(b)$ donde $a \preceq b$ significa que a se usa para construir b.
  $\forall a \preceq b : P(a)$ es la Hipótesis de inducción
- Se concluirá $\forall a | a \in A : P(a)$ por inducción estructural

El principio de inducción estructural (Video 3.8.7)

Sea A un conjunto definido recursivamente.
En general, supongamos que queremos probar

$$\forall a | a \in A : P(a)$$

donde $P(a)$ es un predicado sobre A.
Usaremos el principio de inducción estructural de la siguiente manera:

- Establecer clara y formalmente $P(a)$
- **[Caso base]** Demostrar $P(a)$, para todo $a \in A$ definido por una regla básica, usando las técnicas de demostración conocidas.
- **[Caso de inducción]** Demostrar $P(b)$ para todo $b \in A$ construido con una regla recursiva, suponiendo que todos los elementos $a \in A$ usados en la construcción de b cumplen P. Formalmente podemos escribirlo así: $(\forall a \preceq b : P(a)) \vdash P(b)$ donde $a \preceq b$ significa que a se usa para construir b.
  $\forall a \preceq b : P(a)$ es la Hipótesis de inducción
- Se concluirá $\forall a | a \in A : P(a)$ por inducción estructural

El principio de inducción estructural (Video 3.8.7)

Sea A un conjunto definido recursivamente.
En general, supongamos que queremos probar

$$\forall a | a \in A : P(a)$$

donde $P(a)$ es un predicado sobre A.
Usaremos el principio de inducción estructural de la siguiente manera:

- Establecer clara y formalmente $P(a)$
- **[Caso base]** Demostrar $P(a)$, para todo $a \in A$ definido por una regla básica, usando las técnicas de demostración conocidas.
- **[Caso de inducción]** Demostrar $P(b)$ para todo $b \in A$ construido con una regla recursiva, suponiendo que todos los elementos $a \in A$ usados en la construcción de b cumplen P. Formalmente podemos escribirlo así: $(\forall a \preceq b : P(a)) \vdash P(b)$ donde $a \preceq b$ significa que a se usa para construir b.
  $\forall a \preceq b : P(a)$ es la Hipótesis de inducción
- Se concluirá $\forall a | a \in A : P(a)$ por inducción estructural

El principio de inducción estructural (Video 3.8.7)

Sea A un conjunto definido recursivamente.
En general, supongamos que queremos probar

$$\forall a | a \in A : P(a)$$

donde $P(a)$ es un predicado sobre A.
Usaremos el principio de inducción estructural de la siguiente manera:

- Establecer clara y formalmente $P(a)$
- **[Caso base]** Demostrar $P(a)$, para todo $a \in A$ definido por una regla básica, usando las técnicas de demostración conocidas.
- **[Caso de inducción]** Demostrar $P(b)$ para todo $b \in A$ construido con una regla recursiva, suponiendo que todos los elementos $a \in A$ usados en la construcción de b cumplen P. Formalmente podemos escribirlo así: $(\forall a \preceq b : P(a)) \vdash P(b)$ donde $a \preceq b$ significa que a se usa para construir b.
  $\forall a \preceq b : P(a)$ es la Hipótesis de inducción
- Se concluirá $\forall a | a \in A : P(a)$ por inducción estructural

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x + y) \in A$

Demostrar que

$$\forall a \in A : 3|a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3|a$
- [Caso base] Demostrar $P(3)$
  $P(3)$
- [Caso de inducción] Demostrar $P(x), P(y) \vdash P(x + y)$
- Por tanto, $\forall a \in A : 3|a$

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x+y) \in A$

Demostrar que

$$\forall a \in A : 3|a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3|a$
- **[Caso base]** Demostrar $P(3)$
  $P(3)$
- **[Caso de inducción]** Demostrar $P(x), P(y) \vdash P(x+y)$

- Por tanto, $\forall a \in A : 3|a$

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x + y) \in A$

Demostrar que

$$\forall a \in A : 3|a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3|a$
- **[Caso base]** Demostrar $P(3)$
  $P(3) \equiv 3|3 \equiv true$
- **[Caso de inducción]** Demostrar $P(x), P(y) \vdash P(x + y)$
- Por tanto, $\forall a \in A : 3|a$

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x + y) \in A$

Demostrar que

$$\forall a \in A : 3|a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3|a$
- **[Caso base]** Demostrar $P(3)$
  $P(3) \equiv 3|3 \equiv true$
- **[Caso de inducción]** Demostrar $P(x), P(y) \vdash P(x + y)$
- Por tanto, $\forall a \in A : 3|a$

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x + y) \in A$

Demostrar que

$$\forall a \in A : 3 | a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3 | a$
- **[Caso base]** Demostrar $P(3)$
  $P(3) \equiv 3 | 3 \equiv true$
- **[Caso de inducción]** Demostrar $P(x), P(y) \vdash P(x + y)$

- Por tanto, $\forall a \in A : 3 | a$

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x + y) \in A$

Demostrar que

$$\forall a \in A : 3|a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3|a$
- **[Caso base]** Demostrar $P(3)$
  $P(3) \equiv 3|3 \equiv true$
- **[Caso de inducción]** Demostrar $P(x), P(y) \vdash P(x + y)$

Teo ($P(x+y)$):	$3\|(x+y)$	
Hip. Ind. ($P(x), P(y)$):	$HI : 3\|x \wedge 3\|y$	
	Exp	Just
	$3\|x \wedge 3\|y$	HI
$\implies$	$3\|(x+y)$	Teo. divisibilidad
		$\diamond$

- Por tanto, $\forall a \in A : 3|a$

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x+y) \in A$

Demostrar que

$$\forall a \in A : 3|a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3|a$
- **[Caso base]** Demostrar $P(3)$
  $P(3) \equiv 3|3 \equiv true$
- **[Caso de inducción]** Demostrar $P(x), P(y) \vdash P(x+y)$

Teo ($P(x+y)$):	$3\|(x+y)$	
Hip. Ind. ($P(x), P(y)$):	$HI : 3\|x \wedge 3\|y$	
	Exp.	Just.
	$3\|x \wedge 3\|y$	**HI**
$\implies$	$3\|(x+y)$	**Teo divisibilidad**
		$\diamond$

- Por tanto, $\forall a \in A : 3|a$

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x + y) \in A$

Demostrar que

$$\forall a \in A : 3|a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3|a$
- **[Caso base]** Demostrar $P(3)$
  $P(3) \equiv 3|3 \equiv true$
- **[Caso de inducción]** Demostrar $P(x), P(y) \vdash P(x + y)$

Teo ($P(x+y)$):	$3\|(x+y)$	
Hip. Ind. ($P(x), P(y)$):	$HI : 3\|x \wedge\ 3\|y$	
	Exp.	Just.
	$3\|x \wedge\ 3\|y$	**HI**
$\implies$	$3\|(x+y)$	**Teo divisibilidad**
		$\diamond$

- Por tanto, $\forall a \in A : 3|a$

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x+y) \in A$

Demostrar que

$$\forall a \in A : 3|a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3|a$
- **[Caso base]** Demostrar $P(3)$
  $P(3) \equiv 3|3 \equiv true$
- **[Caso de inducción]** Demostrar $P(x), P(y) \vdash P(x+y)$

Teo ($P(x+y)$):	$3\|(x+y)$	
Hip. Ind. ($P(x), P(y)$):	$HI : 3\|x \wedge \ 3\|y$	
	Exp.	Just.
	$3\|x \wedge \ 3\|y$	**HI**
$\implies$	$3\|(x+y)$	**Teo divisibilidad**
		$\diamond$

- Por tanto, $\forall a \in A : 3|a$

Ejemplo (Video 3.8.8)

Sea A el conjunto definido así:

- $3 \in A$
- $x \in A \wedge y \in A \implies (x+y) \in A$

Demostrar que

$$\forall a \in A : 3|a$$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv 3|a$
- **[Caso base]** Demostrar $P(3)$
  $P(3) \equiv 3|3 \equiv true$
- **[Caso de inducción]** Demostrar $P(x), P(y) \vdash P(x+y)$

Teo ($P(x+y)$):	$3\|(x+y)$	
Hip. Ind. ($P(x), P(y)$):	$HI : 3\|x \wedge 3\|y$	
	Exp.	Just.
	$3\|x \wedge 3\|y$	**HI**
$\implies$	$3\|(x+y)$	**Teo divisibilidad**
		$\diamond$

- Por tanto, $\forall a \in A : 3|a$

Plan

Recordemos sobre las secuencias . . . (Video 3.8.9)

El conjunto $Seq[A]$ se define recursivamente, así:

- $\epsilon \in Seq[A]$
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$

Y las operaciones *size* y $\oplus$ se definen así:

$$size(t) = \begin{cases} 0 & t = \epsilon \\ 1 + size(s) & t = a \triangleleft s \end{cases} \qquad s \oplus t = \begin{cases} t & s = \epsilon \\ a \triangleleft (u \oplus t) & s = a \triangleleft u \end{cases}$$

Vamos a probar las siguientes propiedades:

- $\forall s \in Seq[A] : s \oplus \epsilon = s$
- $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

Recordemos sobre las secuencias . . . (Video 3.8.9)

El conjunto $Seq[A]$ se define recursivamente, así:

- $\epsilon \in Seq[A]$
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$

Y las operaciones *size* y $\oplus$ se definen así:

$$size(t) = \begin{cases} 0 & t = \epsilon \\ 1 + size(s) & t = a \triangleleft s \end{cases} \qquad s \oplus t = \begin{cases} t & s = \epsilon \\ a \triangleleft (u \oplus t) & s = a \triangleleft u \end{cases}$$

Vamos a probar las siguientes propiedades:

- $\forall s \in Seq[A] : s \oplus \epsilon = s$
- $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

Recordemos sobre las secuencias . . . (Video 3.8.9)

El conjunto $Seq[A]$ se define recursivamente, así:

- $\epsilon \in Seq[A]$
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$

Y las operaciones *size* y $\oplus$ se definen así:

$$size(t) = \begin{cases} 0 & t = \epsilon \\ 1 + size(s) & t = a \triangleleft s \end{cases} \qquad s \oplus t = \begin{cases} t & s = \epsilon \\ a \triangleleft (u \oplus t) & s = a \triangleleft u \end{cases}$$

Vamos a probar las siguientes propiedades:

- $\forall s \in Seq[A] : s \oplus \epsilon = s$
- $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

Recordemos sobre las secuencias . . . (Video 3.8.9)

El conjunto $Seq[A]$ se define recursivamente, así:

- $\epsilon \in Seq[A]$
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$

Y las operaciones *size* y $\oplus$ se definen así:

$$size(t) = \begin{cases} 0 & t = \epsilon \\ 1 + size(s) & t = a \triangleleft s \end{cases} \qquad s \oplus t = \begin{cases} t & s = \epsilon \\ a \triangleleft (u \oplus t) & s = a \triangleleft u \end{cases}$$

Vamos a probar las siguientes propiedades:

- $\forall s \in Seq[A] : s \oplus \epsilon = s$
- $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

Recordemos sobre las secuencias ... (Video 3.8.9)

El conjunto $Seq[A]$ se define recursivamente, así:

- $\epsilon \in Seq[A]$
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \lhd s \in Seq[A]$

Y las operaciones *size* y $\oplus$ se definen así:

$$size(t) = \begin{cases} 0 & t = \epsilon \\ 1 + size(s) & t = a \lhd s \end{cases} \qquad s \oplus t = \begin{cases} t & s = \epsilon \\ a \lhd (u \oplus t) & s = a \lhd u \end{cases}$$

Vamos a probar las siguientes propiedades:

- $\forall s \in Seq[A] : s \oplus \epsilon = s$
- $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

Recordemos sobre las secuencias . . . (Video 3.8.9)

El conjunto $Seq[A]$ se define recursivamente, así:

- $\epsilon \in Seq[A]$
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \lhd s \in Seq[A]$

Y las operaciones *size* y $\oplus$ se definen así:

$$size(t) = \begin{cases} 0 & t = \epsilon \\ 1 + size(s) & t = a \lhd s \end{cases} \qquad s \oplus t = \begin{cases} t & s = \epsilon \\ a \lhd (u \oplus t) & s = a \lhd u \end{cases}$$

Vamos a probar las siguientes propiedades:

- $\forall s \in Seq[A] : s \oplus \epsilon = s$
- $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

Recordemos sobre las secuencias . . . (Video 3.8.9)

El conjunto $Seq[A]$ se define recursivamente, así:

- $\epsilon \in Seq[A]$
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$

Y las operaciones *size* y $\oplus$ se definen así:

$$size(t) = \begin{cases} 0 & t = \epsilon \\ 1 + size(s) & t = a \triangleleft s \end{cases} \qquad s \oplus t = \begin{cases} t & s = \epsilon \\ a \triangleleft (u \oplus t) & s = a \triangleleft u \end{cases}$$

Vamos a probar las siguientes propiedades:

- $\forall s \in Seq[A] : s \oplus \epsilon = s$
- $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

Recordemos sobre las secuencias . . . (Video 3.8.9)

El conjunto $Seq[A]$ se define recursivamente, así:

- $\epsilon \in Seq[A]$
- Si $a \in A$ y $s \in Seq[A]$, entonces $t = a \triangleleft s \in Seq[A]$

Y las operaciones $size$ y $\oplus$ se definen así:

$$size(t) = \begin{cases} 0 & t = \epsilon \\ 1 + size(s) & t = a \triangleleft s \end{cases} \qquad s \oplus t = \begin{cases} t & s = \epsilon \\ a \triangleleft (u \oplus t) & s = a \triangleleft u \end{cases}$$

Vamos a probar las siguientes propiedades:

- $\forall s \in Seq[A] : s \oplus \epsilon = s$
- $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s \in Seq[A] : s \oplus \epsilon = s$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural:

- $P(s) \equiv s \oplus \epsilon = s$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv \epsilon \oplus \epsilon = \epsilon \equiv \epsilon = \epsilon \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$
- Por tanto, $\forall s \in Seq[A] : s \oplus \epsilon = s$

$\forall s \in Seq[A] : s \oplus \epsilon = s$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural:

- $P(s) \equiv s \oplus \epsilon = s$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv \epsilon \oplus \epsilon = \epsilon \equiv \epsilon = \epsilon \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$
- Por tanto, $\forall s \in Seq[A] : s \oplus \epsilon = s$

$\forall s \in Seq[A] : s \oplus \epsilon = s$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural:

- $P(s) \equiv s \oplus \epsilon = s$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv \epsilon \oplus \epsilon = \epsilon \equiv \epsilon = \epsilon \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$
- Por tanto, $\forall s \in Seq[A] : s \oplus \epsilon = s$

$\forall s \in Seq[A] : s \oplus \epsilon = s$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural:

- $P(s) \equiv s \oplus \epsilon = s$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv \epsilon \oplus \epsilon = \epsilon \equiv \epsilon = \epsilon \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$

Teo ($P(a \triangleleft s)$):	$a \triangleleft s \oplus \epsilon = a \triangleleft s$	
Hip. Ind. ($P(s)$):	$HI : s \oplus \epsilon = s$	
	Exp.	Just.
	$a \triangleleft s \oplus \epsilon$	
$=$	$a \triangleleft (s \oplus \epsilon)$	Def. $\oplus$
$=$	$a \triangleleft (s)$	HI
		$\diamond$

- Por tanto, $\forall s \in Seq[A] : s \oplus \epsilon = s$

$\forall s \in Seq[A] : s \oplus \epsilon = s$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural:

- $P(s) \equiv s \oplus \epsilon = s$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv \epsilon \oplus \epsilon = \epsilon \equiv \epsilon = \epsilon \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$

Teo ($P(a \triangleleft s)$):	$a \triangleleft s \oplus \epsilon = a \triangleleft s$	
Hip. Ind. ($P(s)$):	$HI : s \oplus \epsilon = s$	
	Exp.	Just.
	$a \triangleleft s \oplus \epsilon$	
$=$	$a \triangleleft (s \oplus \epsilon)$	**Def.** $\oplus$
$=$	$a \triangleleft (s)$	**HI**
		$\diamond$

- Por tanto, $\forall s \in Seq[A] : s \oplus \epsilon = s$

$\forall s \in Seq[A] : s \oplus \epsilon = s$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural:

- $P(s) \equiv s \oplus \epsilon = s$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv \epsilon \oplus \epsilon = \epsilon \equiv \epsilon = \epsilon \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$

Teo ($P(a \triangleleft s)$):	$a \triangleleft s \oplus \epsilon = a \triangleleft s$	
Hip. Ind. ($P(s)$):	$HI : s \oplus \epsilon = s$	
	Exp.	Just.
	$a \triangleleft s \oplus \epsilon$	
$=$	$a \triangleleft (s \oplus \epsilon)$	**Def.** $\oplus$
$=$	$a \triangleleft (s)$	**HI**
		$\diamond$

- Por tanto, $\forall s \in Seq[A] : s \oplus \epsilon = s$

$\forall s \in Seq[A] : s \oplus \epsilon = s$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural:

- $P(s) \equiv s \oplus \epsilon = s$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv \epsilon \oplus \epsilon = \epsilon \equiv \epsilon = \epsilon \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$

Teo ($P(a \triangleleft s)$):	$a \triangleleft s \oplus \epsilon = a \triangleleft s$	
Hip. Ind. ($P(s)$):	$HI : s \oplus \epsilon = s$	
	Exp.	Just.
	$a \triangleleft s \oplus \epsilon$	
$=$	$a \triangleleft (s \oplus \epsilon)$	**Def.** $\oplus$
$=$	$a \triangleleft (s)$	**HI**
		$\diamond$

- Por tanto, $\forall s \in Seq[A] : s \oplus \epsilon = s$

$\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$
- Por tanto, $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \lhd s)$

- Por tanto, $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s,t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$

- Por tanto, $\forall s,t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \lhd s)$
- Por tanto, $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$
- Por tanto, $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$

Teo $(P(a \triangleleft s))$:	$size(a \triangleleft s \oplus t) = size(a \triangleleft s) + size(t)$	
Hip. Ind. $(P(s))$:	$HI : size(s \oplus t) = size(s) + size(t)$	
	Exp.	Just.
	$size(a \triangleleft s \oplus t)$	
$\equiv$	$size(a \triangleleft (s \oplus t))$	Def. $\oplus$
$=$	$1 + size(s \oplus t)$	Def. size
$=$	$1 + size(s) + size(t)$	HI
$=$	$(1 + size(s)) + size(t)$	Aritmética
$=$	$size(a \triangleleft s) + size(t)$	Def. size
		✓

- Por tanto, $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$

Teo ($P(a \triangleleft s)$):	$size(a \triangleleft s \oplus t) = size(a \triangleleft s) + size(t)$	
Hip. Ind. ($P(s)$):	$HI : size(s \oplus t) = size(s) + size(t)$	
	Exp.	Just.
	$size(a \triangleleft s \oplus t)$	
$=$	$size(a \triangleleft (s \oplus t))$	**Def.** $\oplus$
$=$	$1 + size(s \oplus t)$	**Def.** *size*
$=$	$1 + size(s) + size(t)$	**HI**
$=$	$(1 + size(s)) + size(t)$	**Aritmética**
$=$	$size(a \triangleleft s) + size(t)$	**Def.** *size*
		$\diamond$

- Por tanto, $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$

Teo ($P(a \triangleleft s)$):	$size(a \triangleleft s \oplus t) = size(a \triangleleft s) + size(t)$	
Hip. Ind. ($P(s)$):	$HI : size(s \oplus t) = size(s) + size(t)$	
	Exp.	Just.
	$size(a \triangleleft s \oplus t)$	
$=$	$size(a \triangleleft (s \oplus t))$	**Def.** $\oplus$
$=$	$1 + size(s \oplus t)$	**Def.** *size*
$=$	$1 + size(s) + size(t)$	**HI**
$=$	$(1 + size(s)) + size(t)$	**Aritmética**
$=$	$size(a \triangleleft s) + size(t)$	**Def.** *size*
		$\diamond$

- Por tanto, $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \lhd s)$

Teo ($P(a \lhd s)$):	$size(a \lhd s \oplus t) = size(a \lhd s) + size(t)$	
Hip. Ind. ($P(s)$):	$HI : size(s \oplus t) = size(s) + size(t)$	
	Exp.	Just.
	$size(a \lhd s \oplus t)$	
$=$	$size(a \lhd (s \oplus t))$	**Def.** $\oplus$
$=$	$1 + size(s \oplus t)$	**Def.** *size*
$=$	$1 + size(s) + size(t)$	**HI**
$=$	$(1 + size(s)) + size(t)$	**Aritmética**
$=$	$size(a \lhd s) + size(t)$	**Def.** *size*
		$\diamond$

- Por tanto, $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

$\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$ (Video 3.8.9)

Lo probaremos usando el principio de inducción estructural sobre s:

- $P(s) \equiv size(s \oplus t) = size(s) + size(t)$
- **[Caso base]** Demostrar $P(\epsilon)$
  $P(\epsilon) \equiv size(\epsilon \oplus t) = size(\epsilon) + size(t) \equiv size(t) = 0 + size(t)$
  $\equiv size(t) = size(t) \equiv true$
- **[Caso de inducción]** Demostrar $P(s) \vdash P(a \triangleleft s)$

Teo ($P(a \triangleleft s)$):	$size(a \triangleleft s \oplus t) = size(a \triangleleft s) + size(t)$	
Hip. Ind. ($P(s)$):	$HI : size(s \oplus t) = size(s) + size(t)$	
	Exp.	Just.
	$size(a \triangleleft s \oplus t)$	
$=$	$size(a \triangleleft (s \oplus t))$	**Def.** $\oplus$
$=$	$1 + size(s \oplus t)$	**Def.** *size*
$=$	$1 + size(s) + size(t)$	**HI**
$=$	$(1 + size(s)) + size(t)$	**Aritmética**
$=$	$size(a \triangleleft s) + size(t)$	**Def.** *size*
		$\diamond$

- Por tanto, $\forall s, t \in Seq[A] : size(s \oplus t) = size(s) + size(t)$

Plan

Recordemos sobre los árboles binarios completos ...

El conjunto $ArBinC[A]$ se define recursivamente, así:

- Dado $r \in A$, $\nabla(r) \in ArBinC[A]$
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$

Y las operaciones *nodos* y *alt* se definen así:

- $nodos(a) = \begin{cases} 1 & a = \nabla(r) \\ 1 + nodos(a_1) + nodos(a_2) & a = r \rightarrowtail (a_1, a_2) \end{cases}$
- $alt(a) = \begin{cases} 0 & a = \nabla(r) \\ 1 + max(alt(a_1), alt(a_2)) & a = r \rightarrowtail (a_1, a_2) \end{cases}$

Vamos a probar la siguiente propiedad:

$$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$$

Recordemos sobre los árboles binarios completos . . .

El conjunto $ArBinC[A]$ se define recursivamente, así:

- Dado $r \in A$, $\nabla(r) \in ArBinC[A]$
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$

Y las operaciones *nodos* y *alt* se definen así:

- $nodos(a) = \begin{cases} 1 & a = \nabla(r) \\ 1 + nodos(a_1) + nodos(a_2) & a = r \rightarrowtail (a_1, a_2) \end{cases}$
- $alt(a) = \begin{cases} 0 & a = \nabla(r) \\ 1 + max(alt(a_1), alt(a_2)) & a = r \rightarrowtail (a_1, a_2) \end{cases}$

Vamos a probar la siguiente propiedad:

$$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$$

Recordemos sobre los árboles binarios completos . . .

El conjunto $ArBinC[A]$ se define recursivamente, así:

- Dado $r \in A$, $\nabla(r) \in ArBinC[A]$
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$

Y las operaciones *nodos* y *alt* se definen así:

- $nodos(a) = \begin{cases} 1 & a = \nabla(r) \\ 1 + nodos(a_1) + nodos(a_2) & a = r \rightarrowtail (a_1, a_2) \end{cases}$
- $alt(a) = \begin{cases} 0 & a = \nabla(r) \\ 1 + max(alt(a_1), alt(a_2)) & a = r \rightarrowtail (a_1, a_2) \end{cases}$

Vamos a probar la siguiente propiedad:

$$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$$

Recordemos sobre los árboles binarios completos ...

El conjunto $ArBinC[A]$ se define recursivamente, así:

- Dado $r \in A$, $\nabla(r) \in ArBinC[A]$
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$

Y las operaciones *nodos* y *alt* se definen así:

- $nodos(a) = \begin{cases} 1 & a = \nabla(r) \\ 1 + nodos(a_1) + nodos(a_2) & a = r \rightarrowtail (a_1, a_2) \end{cases}$
- $alt(a) = \begin{cases} 0 & a = \nabla(r) \\ 1 + max(alt(a_1), alt(a_2)) & a = r \rightarrowtail (a_1, a_2) \end{cases}$

Vamos a probar la siguiente propiedad:

$$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$$

Recordemos sobre los árboles binarios completos ...

El conjunto $ArBinC[A]$ se define recursivamente, así:

- Dado $r \in A$, $\nabla(r) \in ArBinC[A]$
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$

Y las operaciones *nodos* y *alt* se definen así:

- $nodos(a) = \begin{cases} 1 & a = \nabla(r) \\ 1 + nodos(a_1) + nodos(a_2) & a = r \rightarrowtail (a_1, a_2) \end{cases}$
- $alt(a) = \begin{cases} 0 & a = \nabla(r) \\ 1 + max(alt(a_1), alt(a_2)) & a = r \rightarrowtail (a_1, a_2) \end{cases}$

Vamos a probar la siguiente propiedad:

$$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$$

Recordemos sobre los árboles binarios completos . . .

El conjunto $ArBinC[A]$ se define recursivamente, así:

- Dado $r \in A$, $\nabla(r) \in ArBinC[A]$
- Si $r \in A$ y $a_1, a_2 \in ArBinC[A]$, entonces $a = r \rightarrowtail (a_1, a_2) \in ArBinC[A]$

Y las operaciones *nodos* y *alt* se definen así:

- $nodos(a) = \begin{cases} 1 & a = \nabla(r) \\ 1 + nodos(a_1) + nodos(a_2) & a = r \rightarrowtail (a_1, a_2) \end{cases}$
- $alt(a) = \begin{cases} 0 & a = \nabla(r) \\ 1 + max(alt(a_1), alt(a_2)) & a = r \rightarrowtail (a_1, a_2) \end{cases}$

Vamos a probar la siguiente propiedad:

$$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$$

$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv nodos(a) \leq 2^{alt(a)+1} - 1$
- **[Caso base]** Demostrar $P(\nabla(r))$
  $P(\nabla(r)) \equiv nodos(\nabla(r)) \leq 2^{alt(\nabla(r))+1} - 1 \equiv 1 \leq 2^{0+1} - 1 \equiv 1 \leq 2^1 - 1$
  $\equiv 1 \leq 1 \equiv true$
- **[Caso de inducción]** Demostrar $P(a_1) \wedge P(a_2) \vdash P(r \rightarrowtail (a_1, a_2))$

- Por tanto, $\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv nodos(a) \leq 2^{alt(a)+1} - 1$
- **[Caso base]** Demostrar $P(\nabla(r))$
  $P(\nabla(r)) \equiv nodos(\nabla(r)) \leq 2^{alt(\nabla(r))+1} - 1 \equiv 1 \leq 2^{0+1} - 1 \equiv 1 \leq 2^1 - 1$
  $\equiv 1 \leq 1 \equiv true$
- **[Caso de inducción]** Demostrar $P(a_1) \wedge P(a_2) \vdash P(r \longmapsto (a_1, a_2))$

- Por tanto, $\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv nodos(a) \leq 2^{alt(a)+1} - 1$
- **[Caso base]** Demostrar $P(\nabla(r))$
  $P(\nabla(r)) \equiv nodos(\nabla(r)) \leq 2^{alt(\nabla(r))+1} - 1 \equiv 1 \leq 2^{0+1} - 1 \equiv 1 \leq 2^{1} - 1$
  $\equiv 1 \leq 1 \equiv true$
- **[Caso de inducción]** Demostrar $P(a_1) \wedge P(a_2) \vdash P(r \rightarrow (a_1, a_2))$

- Por tanto, $\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv nodos(a) \leq 2^{alt(a)+1} - 1$
- **[Caso base]** Demostrar $P(\nabla(r))$
  $P(\nabla(r)) \equiv nodos(\nabla(r)) \leq 2^{alt(\nabla(r))+1} - 1 \equiv 1 \leq 2^{0+1} - 1 \equiv 1 \leq 2^1 - 1$
  $\equiv 1 \leq 1 \equiv true$
- **[Caso de inducción]** Demostrar $P(a_1) \wedge P(a_2) \vdash P(r \rightarrowtail (a_1, a_2))$

- Por tanto, $\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv nodos(a) \leq 2^{alt(a)+1} - 1$
- **[Caso base]** Demostrar $P(\nabla(r))$
  $P(\nabla(r)) \equiv nodos(\nabla(r)) \leq 2^{alt(\nabla(r))+1} - 1 \equiv 1 \leq 2^{0+1} - 1 \equiv 1 \leq 2^1 - 1$
  $\equiv 1 \leq 1 \equiv true$
- **[Caso de inducción]** Demostrar $P(a_1) \wedge P(a_2) \vdash P(r \rightarrowtail (a_1, a_2))$

- Por tanto, $\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv nodos(a) \leq 2^{alt(a)+1} - 1$
- **[Caso base]** Demostrar $P(\nabla(r))$
  $P(\nabla(r)) \equiv nodos(\nabla(r)) \leq 2^{alt(\nabla(r))+1} - 1 \equiv 1 \leq 2^{0+1} - 1 \equiv 1 \leq 2^1 - 1$
  $\equiv 1 \leq 1 \equiv true$
- **[Caso de inducción]** Demostrar $P(a_1) \wedge P(a_2) \vdash P(r \rightarrowtail (a_1, a_2))$

Teo ($P(r \rightarrowtail (a_1, a_2))$):	$nodos(r \rightarrowtail (a_1, a_2)) \leq 2^{alt(r \rightarrowtail (a_1, a_2))+1} - 1$	
Hip. Ind. ($P(a_1) \wedge P(a_2)$):	$HI : nodos(a_1) \leq 2^{alt(a_1)+1} - 1 \wedge$ $nodos(a_2) \leq 2^{alt(a_2)+1} - 1$	
	Exp.	Just.
	$nodos(r \rightarrowtail (a_1, a_2))$	
$=$	$1 + nodos(a_1) + nodos(a_2)$	**Def.** *nodos*
$\leq$	$1 + (2^{alt(a_1)+1} - 1) + (2^{alt(a_2)+1} - 1)$	**HI**
$=$	$2^{alt(a_1)+1} + 2^{alt(a_2)+1} - 1$	**Aritmética**
$\leq$	$2max(2^{alt(a_1)+1}, 2^{alt(a_2)+1}) - 1$	$(n+m) \leq 2max(n,m)$
$=$	$2 \times 2^{max(alt(a_1)+1, alt(a_2)+1)} - 1$	$max(2^x, 2^y) = 2^{max(x,y)}$
$=$	$2 \times 2^{max(alt(a_1), alt(a_2))+1} - 1$	$2^{max(x+1,y+1)} = 2^{max(x,y)+1}$
$=$	$2 \times 2^{alt(r \rightarrowtail (a_1, a_2))} - 1$	**Def.** *alt*
$=$	$2^{alt(r \rightarrowtail (a_1, a_2))+1} - 1$	$2 \times 2^x = 2^{x+1}$
		$\diamond$

- Por tanto, $\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv nodos(a) \leq 2^{alt(a)+1} - 1$
- **[Caso base]** Demostrar $P(\nabla(r))$
  $P(\nabla(r)) \equiv nodos(\nabla(r)) \leq 2^{alt(\nabla(r))+1} - 1 \equiv 1 \leq 2^{0+1} - 1 \equiv 1 \leq 2^1 - 1$
  $\equiv 1 \leq 1 \equiv true$
- **[Caso de inducción]** Demostrar $P(a_1) \wedge P(a_2) \vdash P(r \rightarrowtail (a_1, a_2))$

Teo $(P(r \rightarrowtail (a_1, a_2)))$:	$nodos(r \rightarrowtail (a_1, a_2)) \leq 2^{alt(r \rightarrowtail (a_1,a_2))+1} - 1$	
Hip. Ind. $(P(a_1) \wedge P(a_2))$:	$HI : nodos(a_1) \leq 2^{alt(a_1)+1} - 1 \wedge$	
	$nodos(a_2) \leq 2^{alt(a_2)+1} - 1$	
	Exp.	Just.
	$nodos(r \rightarrowtail (a_1, a_2))$	
$=$	$1 + nodos(a_1) + nodos(a_2)$	**Def.** $nodos$
$\leq$	$1 + (2^{alt(a_1)+1} - 1) + (2^{alt(a_2)+1} - 1)$	**HI**
$=$	$2^{alt(a_1)+1} + 2^{alt(a_2)+1} - 1$	**Aritmética**
$\leq$	$2max(2^{alt(a_1)+1}, 2^{alt(a_2)+1}) - 1$	$(n + m) \leq 2max(n, m)$
$=$	$2 \times 2^{max(alt(a_1)+1, alt(a_2)+1)} - 1$	$max(2^x, 2^y) = 2^{max(x,y)}$
$=$	$2 \times 2^{max(alt(a_1), alt(a_2))+1} - 1$	$2^{max(x+1,y+1)} = 2^{max(x,y)+1}$
$=$	$2 \times 2^{alt(r \rightarrowtail (a_1,a_2))} - 1$	**Def.** alt
$=$	$2^{alt(r \rightarrowtail (a_1,a_2))+1} - 1$	$2 \times 2^x = 2^{x+1}$
		$\diamond$

- Por tanto, $\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv nodos(a) \leq 2^{alt(a)+1} - 1$
- **[Caso base]** Demostrar $P(\nabla(r))$
  $P(\nabla(r)) \equiv nodos(\nabla(r)) \leq 2^{alt(\nabla(r))+1} - 1 \equiv 1 \leq 2^{0+1} - 1 \equiv 1 \leq 2^1 - 1$
  $\equiv 1 \leq 1 \equiv true$
- **[Caso de inducción]** Demostrar $P(a_1) \wedge P(a_2) \vdash P(r \rightarrowtail (a_1, a_2))$

Teo ($P(r \rightarrowtail (a_1, a_2))$):	$nodos(r \rightarrowtail (a_1, a_2)) \leq 2^{alt(r \rightarrowtail (a_1,a_2))+1} - 1$	
Hip. Ind. ($P(a_1) \wedge P(a_2)$):	$HI : nodos(a_1) \leq 2^{alt(a_1)+1} - 1 \wedge$ $nodos(a_2) \leq 2^{alt(a_2)+1} - 1$	
	Exp.	Just.
	$nodos(r \rightarrowtail (a_1, a_2))$	
$=$	$1 + nodos(a_1) + nodos(a_2)$	**Def.** *nodos*
$\leq$	$1 + (2^{alt(a_1)+1} - 1) + (2^{alt(a_2)+1} - 1)$	**HI**
$=$	$2^{alt(a_1)+1} + 2^{alt(a_2)+1} - 1$	**Aritmética**
$\leq$	$2max(2^{alt(a_1)+1}, 2^{alt(a_2)+1}) - 1$	$(n+m) \leq 2max(n,m)$
$=$	$2 \times 2^{max(alt(a_1)+1, alt(a_2)+1)} - 1$	$max(2^x, 2^y) = 2^{max(x,y)}$
$=$	$2 \times 2^{max(alt(a_1), alt(a_2))+1} - 1$	$2^{max(x+1,y+1)} = 2^{max(x,y)+1}$
$=$	$2 \times 2^{alt(r \rightarrowtail (a_1,a_2))} - 1$	**Def.** *alt*
$=$	$2^{alt(r \rightarrowtail (a_1,a_2))+1} - 1$	$2 \times 2^x = 2^{x+1}$
		$\diamond$

- Por tanto, $\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

$\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$

Lo probaremos usando el principio de inducción estructural:

- $P(a) \equiv nodos(a) \leq 2^{alt(a)+1} - 1$
- **[Caso base]** Demostrar $P(\nabla(r))$
  $P(\nabla(r)) \equiv nodos(\nabla(r)) \leq 2^{alt(\nabla(r))+1} - 1 \equiv 1 \leq 2^{0+1} - 1 \equiv 1 \leq 2^1 - 1$
  $\equiv 1 \leq 1 \equiv true$
- **[Caso de inducción]** Demostrar $P(a_1) \wedge P(a_2) \vdash P(r \rightarrowtail (a_1, a_2))$

Teo $(P(r \rightarrowtail (a_1, a_2)))$:	$nodos(r \rightarrowtail (a_1, a_2)) \leq 2^{alt(r \rightarrowtail (a_1, a_2))+1} - 1$	
Hip. Ind. $(P(a_1) \wedge P(a_2))$:	$HI : nodos(a_1) \leq 2^{alt(a_1)+1} - 1 \wedge$ $nodos(a_2) \leq 2^{alt(a_2)+1} - 1$	
	Exp.	Just.
	$nodos(r \rightarrowtail (a_1, a_2))$	
$=$	$1 + nodos(a_1) + nodos(a_2)$	**Def.** $nodos$
$\leq$	$1 + (2^{alt(a_1)+1} - 1) + (2^{alt(a_2)+1} - 1)$	**HI**
$=$	$2^{alt(a_1)+1} + 2^{alt(a_2)+1} - 1$	**Aritmética**
$\leq$	$2max(2^{alt(a_1)+1}, 2^{alt(a_2)+1}) - 1$	$(n + m) \leq 2max(n, m)$
$=$	$2 \times 2^{max(alt(a_1)+1, alt(a_2)+1)} - 1$	$max(2^x, 2^y) = 2^{max(x,y)}$
$=$	$2 \times 2^{max(alt(a_1), alt(a_2))+1} - 1$	$2^{max(x+1,y+1)} = 2^{max(x,y)+1}$
$=$	$2 \times 2^{alt(r \rightarrowtail (a_1, a_2))} - 1$	**Def.** alt
$=$	$2^{alt(r \rightarrowtail (a_1, a_2))+1} - 1$	$2 \times 2^x = 2^{x+1}$
		$\diamond$

- Por tanto, $\forall a \in ArBinC[A] : nodos(a) \leq 2^{alt(a)+1} - 1$