Se busca demostrar que ∀`t:: AT a.` ∀`x :: a.` 𝓟(t), donde 𝓟(t) :≡ `elem x (preorder t) = elem x (postorder t).`

Se tienen las siguientes ecuaciones:

```
elem :: Eq a => a -> [a] -> Bool
{E0} elem e [] = False
{E1} elem e (x:xs) = (e == x) || (elem e xs)

preorder :: Procesador (AT a) a
{PR} preorder = foldAT [] (\r rech1 rech2 rech3 -> [r] ++ rech1 ++
rech2 + rech3)

postorder :: Procesador (AT a) a
{PS} postorder = foldAT [] (\r rech1 rech2 rech3 -> rech1 ++ rech2
++ rech3 ++ [r])

foldAT :: b -> (a -> b -> b -> b -> b) -> AT a -> b
{FN} foldAT fNil fTern Nil = fNil
{FT} foldAT fNil fTern (Tern a h1 h2 h3) = fTern a (rec h1) (rec
h2) (rec h3)
     where rec = foldAT fNil fTern
```

`{β} (\x -> Y) Z = Y` reemplazando `x` por `Z`, ∀`Y:: b.` ∀`Z:: a .`

La demostración se realiza por inducción estructural sobre `t::AT a`, donde `data AT a = Nil | Tern r (AT a) (AT a) (AT a)`, de modo que se debe probar lo siguiente:

- **Caso base**: 𝓟(`Nil`) :≡ `elem x (preorder Nil) = elem x (postorder Nil)`
- **Caso inductivo**: ∀`r:: a.` ∀`i:: AT a.` ∀`m:: AT a.` ∀`d:: AT a.` HI ⇒ TI, donde
    - HI :≡ (𝓟(i) ∧ 𝓟(m) ∧ 𝓟(d))
        - 𝓟(i) :≡ `elem x (preorder i) = elem x (postorder i)`
        - 𝓟(m) :≡ `elem x (preorder m) = elem x (postorder m)`
        - 𝓟(d) :≡ `elem x (preorder d) = elem x (postorder d)`
    - TI :≡ `elem x (preorder (Tern r i m d)) = elem x (postorder (Tern r i m d))`

Nota: en las demostraciones que siguen se <u>subrayan</u> los términos que se reemplazarán para pasar a la siguiente expresión, la cual aparece inmediatamente abajo (junto a la/s regla/s usada/s para el reemplazo, entre paréntesis y en color rojo).

El **caso base** se demuestra a continuación:

```
elem x (preorder Nil)
= elem x (foldAT [] (\r rech1 rech2 rech3 -> [r] ++ rech1 ++ rech2
++ rech3) Nil)                                                (por PR)
= elem x []                                                   (por FN)
= elem x (foldAT [] (\r rech1 rech2 rech3 -> rech1 ++ rech2 ++
rech3 ++ [r]) Nil)                                            (por FN)
```

```
= elem x (postorder Nil)                                (por PS)
```

Por otro lado, el **caso inductivo** se demuestra como sigue (usando las abreviaciones `prfTern = (\r rech1 rech2 rech3 -> [r] ++ rech1 ++ rech2 ++rech3)` y `psfTern = (\r rech1 rech2 rech3 -> rech1 ++ rech2 ++ rech3 ++ [r])`):

```
elem x (preorder (Tern r i m d))
= elem x (foldAT [] prfTern (Tern r i m d))                     (por PR)
= elem x (prfTern r (foldAT [] prfTern i) (foldAT [] prfTern m)
(foldAT [] prfTern d))                                          (por FT)
= elem x ([r] ++ (foldAT [] prfTern i) ++ (foldAT [] prfTern m) ++
(foldAT [] prfTern d))                  (por β y definición de prfTern)
= elem x ([r] ++ (preorder i) ++ (preorder m) ++ (preorder d) (por
PR)
= elem x [r] || elem x (preorder i) || elem x (preorder m) || elem
x (preorder d)                                  (por propiedad P0(*))
= elem x (preorder i) || elem x (preorder m) || elem x (preorder
d) || elem x [r]                          (por conmutatividad de ||)
= elem x (postorder i) || elem x (postorder m) || elem x
(postorder d) || elem x [r]                                     (por HI)
= elem x ((postorder i) ++ (postorder m) ++ (postorder d) ++ [r])
                                                (por propiedad P0(*))
= elem x ((foldAT [] psfTern i) ++ (foldAT [] psfTern m) ++
(foldAT [] psfTern d) ++ [r])                                   (por PS)
= elem x (psfTern r (foldAT [] psfTern i) (foldAT [] psfTern m)
(foldAT [] psfTern d))                  (por definición de psfTern y  β)
= elem x (foldAT [] psfTern (Tern r i m d))                     (por FT)
= elem x (postorder (Tern r i m d))                             (por PS)
```

(*): Se puede demostrar la propiedad `P0 :≡ ∀e:: a. ∀xs:: [a]. ∀ys:: [a]. elem e (xs ++ ys) = (elem e xs) || (elem e ys)` por inducción estructural sobre la lista `xs`, usando las siguientes ecuaciones extra:

```
(++) :: [a] -> [a] -> [a]
{A0} [] ++ ys = ys
{A1} (x:xs) ++ ys = x:(xs ++ ys)
```

El **caso base** (`xs = []`) se demuestra como sigue:

```
elem e (xs ++ ys)
= elem e ([] ++ ys)                                     (por xs = [])
= elem e ys                                                  (por A0)
= False || (elem e ys)                           (por False || x = x)
= (elem e []) || (elem e ys)                                 (por E0)
= (elem e xs) || (elem e ys)                            (por xs = [])
```

El **caso inductivo** (HI ⇒ TI, con HI :≡ `elem e (xs ++ ys) = (elem e xs) || (elem e ys)` y TI :≡ `elem e ((x:xs) ++ ys) = (elem e (x:xs)) || (elem e ys)`) se demuestra como sigue:

```
elem e ((x:xs) ++ ys)
= elem e (x:(xs++ys))                                        (por A1)
```

```
= (e == x) || elem e (xs ++ ys)                          (por E1)
= (e == x) || ((elem e xs) || (elem e ys))               (por HI)
= ((e == x) || (elem e xs)) || (elem e ys)   (por asociatividad de
||)
= (elem e (x:xs)) || (elem e ys)                         (por E1)
```