

Traza de reescritura — Heap_H

Aplicación paso a paso de los axiomas a un término concreto, según la especificación *Heap_H.nereus*

Término a evaluar:

```
Heap_DeleteMin(  
  Heap_Insert(  
    Heap_Insert(  
      Heap_Insert(Heap_Empty() as Heap_H, 5),  
      3),  
    7))
```

Idea semántica: insertamos 5, después 3 (que se vuelve raíz porque es menor), después 7 (que va a la rama derecha pero el skew lo cruza a la izquierda); finalmente borramos el mínimo (3) fusionando los dos subárboles.

0. Convenciones

- abrevia `Heap_Empty() as Heap_H` en las vistas de árbol.
- El redex se resalta en **amarillo**.
- Estilo skew heap: cuando recursamos en un subárbol, ese subárbol "rota" al lado opuesto en el resultado.

Traza formal

Paso 1 — caso base de `Heap_Insert` ($i = 5$)

Axioma aplicado:

```
Heap_Insert(Heap_Empty() as Heap_H, i) =  
  BTree_Create(Heap_Empty() as Heap_H,  
    Heap_Empty() as Heap_H,  
    i) as Heap_H;
```

Binding: $i = 5$.

```
Heap_DeleteMin(  
  Heap_Insert(  
    Heap_Insert(  
      BTree_Create(Heap_Empty() as Heap_H, Heap_Empty() as Heap_H, 5) as Heap_H,  
      3),  
    7))
```

Paso 2 — Heap_Insert con $i < x$ ($3 < 5$): 3 sube a la raíz

Axioma aplicado:

```
(i < x) implies
Heap_Insert(BTree_Create(L, R, x) as Heap_H, i) =
  BTree_Create(BTree_Create(L, R, x) as Heap_H,
    Heap_Empty() as Heap_H,
    i) as Heap_H;
```

Bindings: $L = R = \text{Heap_Empty}()$ as Heap_H , $x = 5$, $i = 3$. Condición $3 < 5$ ✓

El heap viejo (singleton 5) queda íntegro como subárbol izquierdo; el derecho queda vacío; 3 es la nueva raíz.

```
Heap_DeleteMin(
  Heap_Insert(
    BTree_Create(
      BTree_Create(·, ·, 5) as Heap_H,
      Heap_Empty() as Heap_H,
      3) as Heap_H,
    7))
```

Después del paso 2

```
  3
 / \
5  ·
/\
· ·
```

Paso 3 — Heap_Insert con not ($i < x$) ($7 \geq 3$): recursa en R + swap

Axioma aplicado:

```
not (i < x) implies
Heap_Insert(BTree_Create(L, R, x) as Heap_H, i) =
  BTree_Create(Heap_Insert(R as Heap_H, i), L, x) as Heap_H;
```

Bindings: $L = \text{BTree_Create}(\cdot, \cdot, 5)$ as Heap_H , $R = \text{Heap_Empty}()$ as Heap_H , $x = 3$, $i = 7$. Condición not ($7 < 3$) ✓

La raíz 3 queda; 7 se inserta en R (que está vacío) y el resultado pasa al lado izquierdo (skew swap). L (el 5) se va a la derecha.

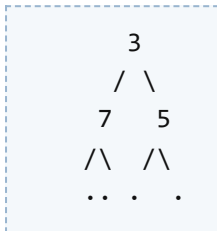
```
Heap_DeleteMin(
  BTree_Create(
    Heap_Insert(Heap_Empty() as Heap_H, 7),
    BTree_Create(·, ·, 5) as Heap_H,
    3) as Heap_H)
```

Paso 4 — caso base de Heap_Insert sobre el subárbol izquierdo nuevo (i = 7)

Mismo axioma del Paso 1, ahora con $i = 7$.

```
Heap_DeleteMin(  
  BTree_Create(  
    BTree_Create(·, ·, 7) as Heap_H,  
    BTree_Create(·, ·, 5) as Heap_H,  
    3) as Heap_H)
```

Heap construido (raíz 3, hijos 7 y 5)



Acá termina la construcción del heap. Notar que 7 quedó como hijo izquierdo de 3 a pesar de haberse insertado "hacia la derecha" — eso es el skew: el subárbol que recibió la recursión termina del lado contrario.

Paso 5 — Heap_DeleteMin: borrar la raíz, fusionar L y R

Axioma aplicado:

```
Heap_DeleteMin(BTree_Create(L, R, x) as Heap_H) =  
  Heap_Merge(L as Heap_H, R as Heap_H);
```

Bindings: $L = \text{BTree_Create}(\cdot, \cdot, 7) \text{ as Heap_H}$, $R = \text{BTree_Create}(\cdot, \cdot, 5) \text{ as Heap_H}$, $x = 3$.

```
Heap_Merge(BTree_Create(·, ·, 7) as Heap_H,  
  BTree_Create(·, ·, 5) as Heap_H)
```

Paso 6 — Heap_Merge con not ($x_1 < x_2$) ($7 \geq 5$): $x_2 = 5$ sube, recursa con R2

Axioma aplicado:

```
not (x1 < x2) implies
Heap_Merge(BTree_Create(L1, R1, x1) as Heap_H,
            BTree_Create(L2, R2, x2) as Heap_H) =
BTree_Create(
    Heap_Merge(BTree_Create(L1, R1, x1) as Heap_H, R2 as Heap_H),
    L2,
    x2
) as Heap_H;
```

Bindings:

L1 = R1 = . ; $x_1 = 7$

L2 = R2 = . ; $x_2 = 5$

Condición: not ($7 < 5$) ✓

Como 5 es la raíz menor, 5 queda arriba. Su subárbol derecho R2 (vacío) se mergea con el primer heap entero (el del 7); el subárbol izquierdo L2 (vacío) queda como nuevo derecho del resultado (skew).

```
BTree_Create(
    Heap_Merge(BTree_Create(., ., 7) as Heap_H, Heap_Empty() as Heap_H),
    Heap_Empty() as Heap_H,
    5
) as Heap_H
```

Paso 7 — Heap_Merge(h, Heap_Empty) = h (identidad)

Axioma aplicado:

```
Heap_Merge(h, Heap_Empty() as Heap_H) = h;
```

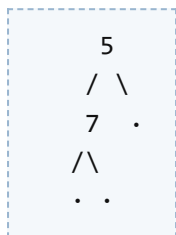
Sobre Heap_Merge(BTree_Create(., ., 7) as Heap_H, Heap_Empty() as Heap_H):

Resultado: BTree_Create(., ., 7) as Heap_H

Forma normal

```
BTree_Create(  
  BTree_Create(Heap_Empty() as Heap_H, Heap_Empty() as Heap_H, 7) as Heap_H,  
  Heap_Empty() as Heap_H,  
  5  
) as Heap_H
```

Vista de árbol del resultado:



Lectura semántica: el heap original contenía $\{3, 5, 7\}$ con 3 a la raíz. `Heap_DeleteMin` sacó el mínimo (3) y fusionó los dos subárboles que tenían 7 y 5. Como $5 < 7$, 5 quedó arriba en el resultado fusionado, con 7 como hijo izquierdo (el skew puso al "ganador del Merge" del lado izquierdo). El nuevo mínimo es 5 — coherente con $\min(\{3, 5, 7\} \setminus \{3\}) = 5$.

Verificación de la propiedad de heap: en el resultado, la raíz 5 es ≤ 7 (su único descendiente). ✓ Invariante preservado, como predice el análisis del Paso 4 de la spec.

Traza generada para la spec `Heap_H.nereus`. Total: 7 pasos (4 para construir el heap, 1 para iniciar el `DeleteMin`, 2 para resolver el Merge interno).