

# Pairing Heap

Hauke Brinkop and Tobias Nipkow

February 6, 2026

## Abstract

This library defines three different versions of pairing heaps: a functional version of the original design based on binary trees [1], the version by Okasaki [2] and a modified version of the latter that is free of structural invariants.

The amortized complexities of these implementations are analyzed in the AFP article [Amortized Complexity](#).

## Contents

<b>1</b>	<b>Pairing Heap in Binary Tree Representation</b>	<b>1</b>
1.1	Definitions . . . . .	2
1.2	Correctness Proofs . . . . .	2
1.2.1	Invariants . . . . .	2
1.2.2	Functional Correctness . . . . .	3
<b>2</b>	<b>Pairing Heap According to Okasaki</b>	<b>4</b>
2.1	Definitions . . . . .	4
2.2	Correctness Proofs . . . . .	5
2.2.1	Invariants . . . . .	5
2.2.2	Functional Correctness . . . . .	6
<b>3</b>	<b>Pairing Heap According to Okasaki (Modified)</b>	<b>7</b>
3.1	Definitions . . . . .	7
3.2	Correctness Proofs . . . . .	8
3.2.1	Invariants . . . . .	8
3.2.2	Functional Correctness . . . . .	9

## 1 Pairing Heap in Binary Tree Representation

```
theory Pairing-Heap-Tree
imports
  HOL-Library.Tree-Multiset
  HOL-Data-Structures.Priority-Queue-Specs
begin
```

## 1.1 Definitions

Pairing heaps [1] in their original representation as binary trees.

```
fun get-min :: 'a :: linorder tree  $\Rightarrow$  'a where  
get-min (Node - x -) = x
```

```
fun link :: ('a::linorder) tree  $\Rightarrow$  'a tree where  
link (Node hsx x (Node hsy y hs)) =  
  (if x < y then Node (Node hsy y hsx) x hs else Node (Node hsx x hsy) y hs) |  
link t = t
```

```
fun pass1 :: ('a::linorder) tree  $\Rightarrow$  'a tree where  
pass1 (Node hsx x (Node hsy y hs)) = link (Node hsx x (Node hsy y (pass1 hs))) |  
pass1 hs = hs
```

```
fun pass2 :: ('a::linorder) tree  $\Rightarrow$  'a tree where  
pass2 (Node hsx x hs) = link(Node hsx x (pass2 hs)) |  
pass2 Leaf = Leaf
```

```
fun del-min :: ('a::linorder) tree  $\Rightarrow$  'a tree where  
del-min Leaf = Leaf  
| del-min (Node hs - -) = pass2 (pass1 hs)
```

```
fun merge :: ('a::linorder) tree  $\Rightarrow$  'a tree  $\Rightarrow$  'a tree where  
merge Leaf hp = hp  
| merge hp Leaf = hp  
| merge (Node hsx x -) (Node hsy y -) = link (Node hsx x (Node hsy y Leaf))
```

Both *del-min* and *merge* need only be defined for arguments that are roots, i.e. of the form  $\langle hp, x, \langle \rangle \rangle$ . For simplicity they are totalized.

```
fun insert :: ('a::linorder)  $\Rightarrow$  'a tree  $\Rightarrow$  'a tree where  
insert x hp = merge (Node Leaf x Leaf) hp
```

The invariant is the conjunction of *is-root* and *pheap*:

```
fun is-root :: 'a tree  $\Rightarrow$  bool where  
is-root hp = (case hp of Leaf  $\Rightarrow$  True | Node l x r  $\Rightarrow$  r = Leaf)
```

```
fun pheap :: ('a :: linorder) tree  $\Rightarrow$  bool where  
pheap Leaf = True |  
pheap (Node l x r) = (( $\forall y \in$  set-tree l. x  $\leq$  y)  $\wedge$  pheap l  $\wedge$  pheap r)
```

## 1.2 Correctness Proofs

### 1.2.1 Invariants

```
lemma link-struct:  $\exists l a.$  link (Node hsx x (Node hsy y hs)) = Node l a hs  
<proof>
```

```
lemma pass1-struct:  $\exists l a r.$  pass1 (Node hs1 x hs) = Node l a r  
<proof>
```

**lemma** *pass2-struct*:  $\exists l a. \text{pass}_2 (\text{Node } hs1 \ x \ hs) = \text{Node } l \ a \ \text{Leaf}$   
(proof)

**lemma** *is-root-merge*:  
 $\text{is-root } h1 \implies \text{is-root } h2 \implies \text{is-root } (\text{merge } h1 \ h2)$   
(proof)

**lemma** *is-root-insert*:  $\text{is-root } h \implies \text{is-root } (\text{insert } x \ h)$   
(proof)

**lemma** *is-root-del-min*:  
**assumes**  $\text{is-root } h$  **shows**  $\text{is-root } (\text{del-min } h)$   
(proof)

**lemma** *pheap-merge*:  
 $\llbracket \text{is-root } h1; \text{is-root } h2; \text{pheap } h1; \text{pheap } h2 \rrbracket \implies \text{pheap } (\text{merge } h1 \ h2)$   
(proof)

**lemma** *pheap-insert*:  $\text{is-root } h \implies \text{pheap } h \implies \text{pheap } (\text{insert } x \ h)$   
(proof)

**lemma** *pheap-link*:  $t \neq \text{Leaf} \implies \text{pheap } t \implies \text{pheap } (\text{link } t)$   
(proof)

**lemma** *pheap-pass1*:  $\text{pheap } h \implies \text{pheap } (\text{pass}_1 \ h)$   
(proof)

**lemma** *pheap-pass2*:  $\text{pheap } h \implies \text{pheap } (\text{pass}_2 \ h)$   
(proof)

**lemma** *pheap-del-min*:  $\text{is-root } h \implies \text{pheap } h \implies \text{pheap } (\text{del-min } h)$   
(proof)

## 1.2.2 Functional Correctness

**lemma** *get-min-in*:  
 $h \neq \text{Leaf} \implies \text{get-min } h \in \text{set-tree } h$   
(proof)

**lemma** *get-min-min*:  $\llbracket \text{is-root } h; \text{pheap } h; x \in \text{set-tree } h \rrbracket \implies \text{get-min } h \leq x$   
(proof)

**lemma** *mset-link*:  $\text{mset-tree } (\text{link } t) = \text{mset-tree } t$   
(proof)

**lemma** *mset-pass1*:  $\text{mset-tree } (\text{pass}_1 \ h) = \text{mset-tree } h$   
(proof)

**lemma** *mset-pass2*:  $mset-tree (pass_2 h) = mset-tree h$   
*<proof>*

**lemma** *mset-merge*:  $\llbracket is-root h1; is-root h2 \rrbracket$   
 $\implies mset-tree (merge h1 h2) = mset-tree h1 + mset-tree h2$   
*<proof>*

**lemma** *mset-del-min*:  $\llbracket is-root h; t \neq Leaf \rrbracket \implies$   
 $mset-tree (del-min h) = mset-tree h - \{\#get-min h\# \}$   
*<proof>*

Last step: prove all axioms of the priority queue specification:

**interpretation** *pairing*: *Priority-Queue-Merge*  
**where** *empty* = *Leaf* **and** *is-empty* =  $\lambda h. h = Leaf$   
**and** *merge* = *merge* **and** *insert* = *insert*  
**and** *del-min* = *del-min* **and** *get-min* = *get-min*  
**and** *invar* =  $\lambda h. is-root h \wedge pheap h$  **and** *mset* = *mset-tree*  
*<proof>*

**end**

## 2 Pairing Heap According to Okasaki

**theory** *Pairing-Heap-List1*  
**imports**  
  *HOL-Library.Multiset*  
  *HOL-Library.Pattern-Aliases*  
  *HOL-Data-Structures.Priority-Queue-Specs*  
**begin**

### 2.1 Definitions

This implementation follows Okasaki [2]. It satisfies the invariant that *Empty* only occurs at the root of a pairing heap. The functional correctness proof does not require the invariant but the amortized analysis (elsewhere) makes use of it.

**datatype** *'a heap* = *Empty* | *Hp 'a 'a heap list*

**fun** *get-min* :: *'a heap*  $\Rightarrow$  *'a* **where**  
*get-min* (*Hp* *x* *-*) = *x*

**hide-const** (**open**) *insert*

**context** **includes** *pattern-aliases*  
**begin**

**fun** *merge* :: (*'a::linorder*) *heap*  $\Rightarrow$  *'a heap*  $\Rightarrow$  *'a heap* **where**

```

merge h Empty = h |
merge Empty h = h |
merge (Hp x hsx =: hx) (Hp y hsy =: hy) =
  (if x < y then Hp x (hy # hsx) else Hp y (hx # hsy))

```

**end**

```

fun insert :: ('a::linorder) => 'a heap => 'a heap where
insert x h = merge (Hp x []) h

```

```

fun pass1 :: ('a::linorder) heap list => 'a heap list where
pass1 (h1#h2#hs) = merge h1 h2 # pass1 hs |
pass1 hs = hs

```

```

fun pass2 :: ('a::linorder) heap list => 'a heap where
pass2 [] = Empty
| pass2 (h#hs) = merge h (pass2 hs)

```

```

fun merge-pairs :: ('a::linorder) heap list => 'a heap where
merge-pairs [] = Empty
| merge-pairs [h] = h
| merge-pairs (h1 # h2 # hs) = merge (merge h1 h2) (merge-pairs hs)

```

```

fun del-min :: ('a::linorder) heap => 'a heap where
del-min Empty = Empty
| del-min (Hp x hs) = pass2 (pass1 hs)

```

## 2.2 Correctness Proofs

An optimization:

```

lemma pass12-merge-pairs: pass2 (pass1 hs) = merge-pairs hs
<proof>

```

```

declare pass12-merge-pairs[code-unfold]

```

### 2.2.1 Invariants

```

fun mset-heap :: 'a heap => 'a multiset where
mset-heap Empty = {#} |
mset-heap (Hp x hs) = {#x#} + sum-mset(mset(map mset-heap hs))

```

```

fun pheap :: ('a :: linorder) heap => bool where
pheap Empty = True |
pheap (Hp x hs) = (∀ h ∈ set hs. (∀ y ∈# mset-heap h. x ≤ y) ∧ pheap h)

```

```

lemma pheap-merge: pheap h1 ==> pheap h2 ==> pheap (merge h1 h2)
<proof>

```

```

lemma pheap-merge-pairs: ∀ h ∈ set hs. pheap h ==> pheap (merge-pairs hs)

```

*<proof>*

**lemma** *pheap-insert*:  $pheap\ h \implies pheap\ (insert\ x\ h)$   
*<proof>*

**lemma** *pheap-del-min*:  $pheap\ h \implies pheap\ (del-min\ h)$   
*<proof>*

## 2.2.2 Functional Correctness

**lemma** *mset-heap-empty-iff*:  $mset-heap\ h = \{\#\} \longleftrightarrow h = Empty$   
*<proof>*

**lemma** *get-min-in*:  $h \neq Empty \implies get-min\ h \in\# mset-heap(h)$   
*<proof>*

**lemma** *get-min-min*:  $\llbracket h \neq Empty; pheap\ h; x \in\# mset-heap(h) \rrbracket \implies get-min\ h \leq x$   
*<proof>*

**lemma** *get-min*:  $\llbracket pheap\ h; h \neq Empty \rrbracket \implies get-min\ h = Min-mset\ (mset-heap\ h)$   
*<proof>*

**lemma** *mset-merge*:  $mset-heap\ (merge\ h1\ h2) = mset-heap\ h1 + mset-heap\ h2$   
*<proof>*

**lemma** *mset-insert*:  $mset-heap\ (insert\ a\ h) = \{\#a\#\} + mset-heap\ h$   
*<proof>*

**lemma** *mset-merge-pairs*:  $mset-heap\ (merge-pairs\ hs) = sum-mset(image-mset\ mset-heap(mset\ hs))$   
*<proof>*

**lemma** *mset-del-min*:  $h \neq Empty \implies mset-heap\ (del-min\ h) = mset-heap\ h - \{\#get-min\ h\#\}$   
*<proof>*

Last step: prove all axioms of the priority queue specification:

**interpretation** *pairing*: *Priority-Queue-Merge*  
**where** *empty* = *Empty* **and** *is-empty* =  $\lambda h. h = Empty$   
**and** *merge* = *merge* **and** *insert* = *insert*  
**and** *del-min* = *del-min* **and** *get-min* = *get-min*  
**and** *invar* = *pheap* **and** *mset* = *mset-heap*  
*<proof>*

**end**

### 3 Pairing Heap According to Oksaki (Modified)

```
theory Pairing-Heap-List2
imports
  HOL-Library.Multiset
  HOL-Data-Structures.Priority-Queue-Specs
begin
```

#### 3.1 Definitions

This version of pairing heaps is a modified version of the one by Okasaki [2] that avoids structural invariants.

```
datatype 'a hp = Hp 'a (hps: 'a hp list)
```

```
type-synonym 'a heap = 'a hp option
```

```
hide-const (open) insert
```

```
fun get-min :: 'a heap  $\Rightarrow$  'a where
get-min (Some(Hp x -)) = x
```

```
fun link :: ('a::linorder) hp  $\Rightarrow$  'a hp  $\Rightarrow$  'a hp where
link (Hp x1 hs1) (Hp x2 hs2) =
  (if x1 < x2 then Hp x1 (Hp x2 hs2 # hs1) else Hp x2 (Hp x1 hs1 # hs2))
```

```
fun merge :: ('a::linorder) heap  $\Rightarrow$  'a heap  $\Rightarrow$  'a heap where
merge ho None = ho |
merge None ho = ho |
merge (Some h1) (Some h2) = Some(link h1 h2)
```

```
lemma merge-None[simp]: merge None ho = ho
<proof>
```

```
fun insert :: ('a::linorder)  $\Rightarrow$  'a heap  $\Rightarrow$  'a heap where
insert x None = Some(Hp x []) |
insert x (Some h) = Some(link (Hp x []) h)
```

```
fun pass1 :: ('a::linorder) hp list  $\Rightarrow$  'a hp list where
pass1 (h1 # h2 # hs) = link h1 h2 # pass1 hs |
pass1 hs = hs
```

```
fun pass2 :: ('a::linorder) hp list  $\Rightarrow$  'a heap where
pass2 [] = None |
pass2 (h # hs) = Some(case pass2 hs of None  $\Rightarrow$  h | Some h'  $\Rightarrow$  link h h')
```

```
fun merge-pairs :: ('a::linorder) hp list  $\Rightarrow$  'a heap where
merge-pairs [] = None
| merge-pairs [h] = Some h
```

| *merge-pairs* (*h1* # *h2* # *hs*) =  
*Some*(*let* *h12* = *link* *h1* *h2* *in case merge-pairs* *hs* *of None*  $\Rightarrow$  *h12* | *Some* *h*  $\Rightarrow$   
*link* *h12* *h*)

**fun** *del-min* :: ('a::linorder) heap  $\Rightarrow$  'a heap **where**  
*del-min* *None* = *None*  
| *del-min* (*Some*(*Hp* *x* *hs*)) = *pass*<sub>2</sub> (*pass*<sub>1</sub> *hs*)

## 3.2 Correctness Proofs

An optimization:

**lemma** *pass12-merge-pairs*: *pass*<sub>2</sub> (*pass*<sub>1</sub> *hs*) = *merge-pairs* *hs*  
<proof>

**declare** *pass12-merge-pairs*[*code-unfold*]

Abstraction functions:

**fun** *mset-hp* :: 'a hp  $\Rightarrow$  'a multiset **where**  
*mset-hp* (*Hp* *x* *hs*) = {#*x*#} + *sum-list*(*map* *mset-hp* *hs*)

**definition** *mset-heap* :: 'a heap  $\Rightarrow$  'a multiset **where**  
*mset-heap* *ho* = (*case* *ho* *of None*  $\Rightarrow$  {#} | *Some* *h*  $\Rightarrow$  *mset-hp* *h*)

### 3.2.1 Invariants

**fun** *php* :: ('a::linorder) hp  $\Rightarrow$  bool **where**  
*php* (*Hp* *x* *hs*) = ( $\forall$  *h*  $\in$  *set* *hs*. ( $\forall$  *y*  $\in$  # *mset-hp* *h*. *x*  $\leq$  *y*)  $\wedge$  *php* *h*)

**definition** *invar* :: ('a::linorder) heap  $\Rightarrow$  bool **where**  
*invar* *ho* = (*case* *ho* *of None*  $\Rightarrow$  *True* | *Some* *h*  $\Rightarrow$  *php* *h*)

**lemma** *php-link*: *php* *h1*  $\Longrightarrow$  *php* *h2*  $\Longrightarrow$  *php* (*link* *h1* *h2*)  
<proof>

**lemma** *invar-merge*:  
[[ *invar* *ho1*; *invar* *ho2* ]]  $\Longrightarrow$  *invar* (*merge* *ho1* *ho2*)  
<proof>

**lemma** *invar-insert*: *invar* *ho*  $\Longrightarrow$  *invar* (*insert* *x* *ho*)  
<proof>

**lemma** *invar-pass1*:  $\forall$  *h*  $\in$  *set* *hs*. *php* *h*  $\Longrightarrow$   $\forall$  *h*  $\in$  *set* (*pass*<sub>1</sub> *hs*). *php* *h*  
<proof>

**lemma** *invar-pass2*:  $\forall$  *h*  $\in$  *set* *hs*. *php* *h*  $\Longrightarrow$  *invar* (*pass*<sub>2</sub> *hs*)  
<proof>

**lemma** *invar-Some*: *invar*(*Some* *h*) = *php* *h*  
<proof>

**lemma** *invar-del-min*:  $\text{invar } ho \implies \text{invar } (\text{del-min } ho)$   
(proof)

### 3.2.2 Functional Correctness

**lemma** *mset-hp-empty[simp]*:  $\text{mset-hp } h \neq \{\#\}$   
(proof)

**lemma** *mset-heap-Some*:  $\text{mset-heap}(\text{Some } h) = \text{mset-hp } h$   
(proof)

**lemma** *mset-heap-empty*:  $\text{mset-heap } h = \{\#\} \iff h = \text{None}$   
(proof)

**lemma** *get-min-in*:  
 $ho \neq \text{None} \implies \text{get-min } ho \in\# \text{ mset-hp}(\text{the } ho)$   
(proof)

**lemma** *get-min-min*:  $\llbracket ho \neq \text{None}; \text{invar } ho; x \in\# \text{ mset-hp}(\text{the } ho) \rrbracket \implies \text{get-min } ho \leq x$   
(proof)

**lemma** *mset-link*:  $\text{mset-hp } (\text{link } h1 \ h2) = \text{mset-hp } h1 + \text{mset-hp } h2$   
(proof)

**lemma** *mset-merge*:  $\text{mset-heap } (\text{merge } ho1 \ ho2) = \text{mset-heap } ho1 + \text{mset-heap } ho2$   
(proof)

**lemma** *mset-insert*:  $\text{mset-heap } (\text{insert } a \ ho) = \{\#a\#\} + \text{mset-heap } ho$   
(proof)

**lemma** *mset-pass<sub>1</sub>*:  $\text{sum-list}(\text{map } \text{mset-hp } (\text{pass}_1 \ hs)) = \text{sum-list}(\text{map } \text{mset-hp } hs)$   
(proof)

**lemma** *mset-pass<sub>2</sub>*:  $\text{mset-heap } (\text{pass}_2 \ hs) = \text{sum-list}(\text{map } \text{mset-hp } hs)$   
(proof)

**lemma** *mset-del-min*:  $ho \neq \text{None} \implies$   
 $\text{mset-heap } (\text{del-min } ho) = \text{mset-heap } ho - \{\#\text{get-min } ho\#\}$   
(proof)

Last step: prove all axioms of the priority queue specification:

**interpretation** *pairing*: *Priority-Queue-Merge*  
**where** *empty* = *None* **and** *is-empty* =  $\lambda h. h = \text{None}$   
**and** *merge* = *merge* **and** *insert* = *insert*  
**and** *del-min* = *del-min* **and** *get-min* = *get-min*  
**and** *invar* = *invar* **and** *mset* = *mset-heap*  
(proof)

**end**

## **References**

- [1] M. L. Fredman, R. Sedgewick, D. D. Sleator, and R. E. Tarjan. The pairing heap: A new form of self-adjusting heap. *Algorithmica*, 1(1):111–129, 1986.
- [2] C. Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.