Pairing Heap

Hauke Brinkop and Tobias Nipkow

March 17, 2025

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

1 Pairing Heap in Binary Tree Representation **1**
1.1 Definitions . . . 2
1.2 Correctness Proofs . . . 2
1.2.1 Invariants . . . 2
1.2.2 Functional Correctness . . . 3

2 Pairing Heap According to Okasaki **5**
2.1 Definitions . . . 5
2.2 Correctness Proofs . . . 6
2.2.1 Invariants . . . 6
2.2.2 Functional Correctness . . . 6

3 Pairing Heap According to Oksaki (Modified) **8**
3.1 Definitions . . . 8
3.2 Correctness Proofs . . . 9
3.2.1 Invariants . . . 9
3.2.2 Functional Correctness . . . 10

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 $pass_1$ :: ($'a$::*linorder*) *tree* $\Rightarrow$ $'a$ *tree* **where**
$pass_1$ (*Node hsx x* (*Node hsy y hs*)) = *link* (*Node hsx x* (*Node hsy y* ($pass_1$ *hs*))) |
$pass_1$ *hs* = *hs*

fun $pass_2$ :: ($'a$::*linorder*) *tree* $\Rightarrow$ $'a$ *tree* **where**
$pass_2$ (*Node hsx x hs*) = *link*(*Node hsx x* ($pass_2$ *hs*)) |
$pass_2$ *Leaf* = *Leaf*

fun *del-min* :: ($'a$::*linorder*) *tree* $\Rightarrow$ $'a$ *tree* **where**
  del-min Leaf = *Leaf*
| *del-min* (*Node hs* - -) = $pass_2$ ($pass_1$ *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*
by *simp*

lemma $pass_1$*-struct*: $\exists\, l\ a\ r$. $pass_1$ (*Node hs1 x hs*) = *Node l a r*
by (*cases hs*) *simp-all*

lemma *pass*$_2$*-struct*: $\exists\, l\ a.\ pass_2\ (Node\ hs1\ x\ hs) = Node\ l\ a\ Leaf$
by(*induction hs arbitrary*: *hs1 x rule*: *pass*$_2$*.induct*) (*auto*, *metis link-struct*)

lemma *is-root-merge*:
  is-root h1 $\Longrightarrow$ *is-root h2* $\Longrightarrow$ *is-root* (*merge h1 h2*)
by (*simp split*: *tree.splits*)

lemma *is-root-insert*: *is-root h* $\Longrightarrow$ *is-root* (*insert x h*)
by (*simp split*: *tree.splits*)

lemma *is-root-del-min*:
  assumes *is-root h* **shows** *is-root* (*del-min h*)
proof (*cases h*)
  case [*simp*]: (*Node lx x rx*)
  have *rx* = *Leaf* **using** *assms* **by** *simp*
  thus *?thesis*
  proof (*cases lx*)
    case (*Node ly y ry*)
    then obtain *la a ra* **where** $pass_1\ lx = Node\ a\ la\ ra$
      using *pass*$_1$*-struct* **by** *blast*
    moreover obtain *lb b* **where** $pass_2\ \ldots = Node\ b\ lb\ Leaf$
      using *pass*$_2$*-struct* **by** *blast*
    ultimately show *?thesis* **using** *assms* **by** *simp*
  qed *simp*
qed *simp*

lemma *pheap-merge*:
  $[\![$ *is-root h1*; *is-root h2*; *pheap h1*; *pheap h2* $]\!]$ $\Longrightarrow$ *pheap* (*merge h1 h2*)
by (*auto split*: *tree.splits*)

lemma *pheap-insert*: *is-root h* $\Longrightarrow$ *pheap h* $\Longrightarrow$ *pheap* (*insert x h*)
by (*auto split*: *tree.splits*)

lemma *pheap-link*: $t \neq Leaf$ $\Longrightarrow$ *pheap t* $\Longrightarrow$ *pheap* (*link t*)
by(*induction t rule*: *link.induct*)(*auto*)

lemma *pheap-pass1*: *pheap h* $\Longrightarrow$ *pheap* ($pass_1\ h$)
by(*induction h rule*: *pass*$_1$*.induct*) (*auto*)

lemma *pheap-pass2*: *pheap h* $\Longrightarrow$ *pheap* ($pass_2\ h$)
by (*induction h*)(*auto simp*: *pheap-link*)

lemma *pheap-del-min*: *is-root h* $\Longrightarrow$ *pheap h* $\Longrightarrow$ *pheap* (*del-min h*)
by (*auto simp*: *pheap-pass1 pheap-pass2 split*: *tree.splits*)

1.2.2 Functional Correctness

lemma *get-min-in*:

$h \neq Leaf \Longrightarrow get\text{-}min\ h \in set\text{-}tree\ h$
by(*auto simp add*: *neq-Leaf-iff*)

lemma *get-min-min*: $[\![$ *is-root h*; *pheap h*; $x \in set\text{-}tree\ h$ $]\!] \Longrightarrow get\text{-}min\ h \leq x$
by(*auto split*: *tree.splits*)

lemma *mset-link*: *mset-tree* (*link t*) = *mset-tree t*
by(*cases t rule*: *link.cases*)(*auto simp*: *add-ac*)

lemma *mset-pass*$_1$: *mset-tree* (*pass*$_1$ *h*) = *mset-tree h*
by (*induction h rule*: *pass*$_1$*.induct*) *auto*

lemma *mset-pass*$_2$: *mset-tree* (*pass*$_2$ *h*) = *mset-tree h*
by (*induction h rule*: *pass*$_2$*.induct*) (*auto simp*: *mset-link*)

lemma *mset-merge*: $[\![$ *is-root h1*; *is-root h2* $]\!]$
$\Longrightarrow$ *mset-tree* (*merge h1 h2*) = *mset-tree h1* + *mset-tree h2*
by (*induction h1 h2 rule*: *merge.induct*) (*auto simp add*: *ac-simps*)

lemma *mset-del-min*: $[\![$ *is-root h*; $t \neq Leaf$ $]\!] \Longrightarrow$
mset-tree (*del-min h*) = *mset-tree h* − {#*get-min h*#}
by(*induction h rule*: *del-min.induct*)(*auto simp*: *mset-pass*$_1$ *mset-pass*$_2$)

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(*standard*, *goal-cases*)
case *1* **show** *?case* **by** *simp*
next
case (*2 q*) **show** *?case* **by** (*cases q*) *auto*
next
case *3* **thus** *?case* **by**(*simp add*: *mset-merge*)
next
case *4* **thus** *?case* **by**(*simp add*: *mset-del-min*)
next
case *5* **thus** *?case* **by**(*simp add*: *eq-Min-iff get-min-in get-min-min*)
next
case *6* **thus** *?case* **by**(*simp*)
next
case *7* **thus** *?case* **using** *is-root-insert pheap-insert* **by** *blast*
next
case *8* **thus** *?case* **using** *is-root-del-min pheap-del-min* **by** *blast*
next
case *9* **thus** *?case* **by** (*simp add*: *mset-merge*)
next

case *10* **thus** *?case* **using** *is-root-merge pheap-merge* **by** *blast*
qed

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 ⇒ 'a* **where**
get-min (Hp x -) = x

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

context includes *pattern-aliases*
begin

fun *merge* :: *('a::linorder) heap ⇒ 'a heap ⇒ '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 $pass_1$ :: *('a::linorder) heap list ⇒ 'a heap list* **where**
$pass_1$ *(h1#h2#hs) = merge h1 h2 #* $pass_1$ *hs |*
$pass_1$ *hs = hs*

fun $pass_2$ :: *('a::linorder) heap list ⇒ 'a heap* **where**
$pass_2$ *[] = Empty*
| $pass_2$ *(h#hs) = merge h (*$pass_2$ *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*) = $pass_2$ ($pass_1$ *hs*)

2.2 Correctness Proofs

An optimization:

lemma *pass12-merge-pairs*: $pass_2$ ($pass_1$ *hs*) = *merge-pairs hs*
by (*induction hs rule*: *merge-pairs.induct*) (*auto split*: *option.split*)

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*)
by (*induction h1 h2 rule*: *merge.induct*) *fastforce*+

lemma *pheap-merge-pairs*: ∀ *h* ∈ *set hs*. *pheap h* ⟹ *pheap* (*merge-pairs hs*)
by (*induction hs rule*: *merge-pairs.induct*)(*auto simp*: *pheap-merge*)

lemma *pheap-insert*: *pheap h* ⟹ *pheap* (*insert x h*)
by (*auto simp*: *pheap-merge*)

lemma *pheap-del-min*: *pheap h* ⟹ *pheap* (*del-min h*)
by(*cases h*) (*auto simp*: *pass12-merge-pairs pheap-merge-pairs*)

2.2.2 Functional Correctness

lemma *mset-heap-empty-iff*: *mset-heap h* = {#} ⟷ *h* = *Empty*
by (*cases h*) *auto*

lemma *get-min-in*: *h* ≠ *Empty* ⟹ *get-min h* ∈# *mset-heap*(*h*)
by(*induction rule*: *get-min.induct*)(*auto*)

lemma *get-min-min*: $[\![\ h \neq Empty;\ pheap\ h;\ x \in\#\ mset\text{-}heap(h)\ ]\!] \Longrightarrow get\text{-}min\ h \leq x$
by(*induction h rule*: *get-min.induct*)(*auto*)

lemma *get-min*: $[\![\ pheap\ h;\ \ h \neq Empty\ ]\!] \Longrightarrow get\text{-}min\ h = Min\text{-}mset\ (mset\text{-}heap\ h)$
by (*metis Min-eqI finite-set-mset get-min-in get-min-min*)

lemma *mset-merge*: *mset-heap* (*merge h1 h2*) = *mset-heap h1* + *mset-heap h2*
by(*induction h1 h2 rule*: *merge.induct*)(*auto simp*: *add-ac*)

lemma *mset-insert*: *mset-heap* (*insert a h*) = {#*a*#} + *mset-heap h*
by(*cases h*) (*auto simp add*: *mset-merge insert-def add-ac*)

lemma *mset-merge-pairs*: *mset-heap* (*merge-pairs hs*) = *sum-mset*(*image-mset mset-heap*(*mset hs*))
by(*induction hs rule*: *merge-pairs.induct*)(*auto simp*: *mset-merge*)

lemma *mset-del-min*: $h \neq Empty \Longrightarrow$
mset-heap (*del-min h*) = *mset-heap h* − {#*get-min h*#}
by(*cases h*) (*auto simp*: *pass12-merge-pairs mset-merge-pairs*)

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(*standard*, *goal-cases*)
  case *1* **show** *?case* **by** *simp*
next
  case (*2 q*) **show** *?case* **by** (*cases q*) *auto*
next
  case *3* **show** *?case* **by**(*simp add*: *mset-insert mset-merge*)
next
  case *4* **thus** *?case* **by**(*simp add*: *mset-del-min mset-heap-empty-iff*)
next
  case *5* **thus** *?case* **using** *get-min mset-heap.simps*(*1*) **by** *blast*
next
  case *6* **thus** *?case* **by**(*simp*)
next
  case *7* **thus** *?case* **by**(*rule pheap-insert*)
next
  case *8* **thus** *?case* **by** (*simp add*: *pheap-del-min*)
next
  case *9* **thus** *?case* **by** (*simp add*: *mset-merge*)
next
  case *10* **thus** *?case* **by** (*simp add*: *pheap-merge*)
qed

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 ⇒ 'a* **where**
get-min (Some(Hp x -)) = x

fun *link* :: *('a::linorder) hp ⇒ 'a hp ⇒ '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 ⇒ 'a heap ⇒ '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*
by(*cases ho*)*auto*

fun *insert* :: *('a::linorder) ⇒ 'a heap ⇒ 'a heap* **where**
insert x None = Some(Hp x []) |
insert x (Some h) = Some(link (Hp x []) h)

fun $pass_1$ :: *('a::linorder) hp list ⇒ 'a hp list* **where**
$pass_1$ *(h1#h2#hs) = link h1 h2 #* $pass_1$ *hs |*
$pass_1$ *hs = hs*

fun $pass_2$ :: *('a::linorder) hp list ⇒ 'a heap* **where**
$pass_2$ *[] = None |*
$pass_2$ *(h#hs) = Some(case* $pass_2$ *hs of None ⇒ h | Some h' ⇒ link h h')*

fun *merge-pairs* :: (*'a*::*linorder*) *hp list* ⇒ *'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* ⇒ *h12* | *Some h* ⇒ *link h12 h*)

fun *del-min* :: (*'a*::*linorder*) *heap* ⇒ *'a heap* **where**
del-min None = *None*
| *del-min* (*Some*(*Hp x hs*)) = $pass_2$ ($pass_1$ *hs*)

3.2 Correctness Proofs

An optimization:

lemma *pass12-merge-pairs*: $pass_2$ ($pass_1$ *hs*) = *merge-pairs hs*
by (*induction hs rule*: *merge-pairs.induct*) (*auto split*: *option.split*)

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

Abstraction functions:

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

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

3.2.1 Invariants

fun *php* :: (*'a*::*linorder*) *hp* ⇒ *bool* **where**
php (*Hp x hs*) = (∀ *h* ∈ *set hs*. (∀ *y* ∈# *mset-hp h*. *x* ≤ *y*) ∧ *php h*)

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

lemma *php-link*: *php h1* ⟹ *php h2* ⟹ *php* (*link h1 h2*)
by (*induction h1 h2 rule*: *link.induct*) (*fastforce simp flip*: *sum-mset-sum-list*)+

lemma *invar-merge*:
⟦ *invar ho1*; *invar ho2* ⟧ ⟹ *invar* (*merge ho1 ho2*)
by (*auto simp*: *php-link invar-def split*: *option.splits*)

lemma *invar-insert*: *invar ho* ⟹ *invar* (*insert x ho*)
by (*auto simp*: *php-link invar-def split*: *option.splits*)

lemma *invar-pass1*: ∀ *h* ∈ *set hs*. *php h* ⟹ ∀ *h* ∈ *set* ($pass_1$ *hs*). *php h*
by(*induction hs rule*: $pass_1$*.induct*) (*auto simp*: *php-link*)

lemma *invar-pass2*: ∀ *h* ∈ *set hs*. *php h* ⟹ *invar* ($pass_2$ *hs*)

by (*induction hs*)(*auto simp*: *php-link invar-def split*: *option.splits*)

lemma *invar-Some*: $invar(Some\ h) = php\ h$
by(*simp add*: *invar-def*)

lemma *invar-del-min*: $invar\ ho \Longrightarrow invar\ (del\text{-}min\ ho)$
by(*induction ho rule*: *del-min.induct*)
(*auto simp*: *invar-Some intro*!: *invar-pass1 invar-pass2*)

3.2.2 Functional Correctness

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

lemma *mset-heap-Some*: $mset\text{-}heap(Some\ h) = mset\text{-}hp\ h$
by(*simp add*: *mset-heap-def*)

lemma *mset-heap-empty*: $mset\text{-}heap\ h = \{\#\} \longleftrightarrow h = None$
by (*cases h*) (*auto simp add*: *mset-heap-def*)

lemma *get-min-in*:
$ho \neq None \Longrightarrow get\text{-}min\ ho \in\# mset\text{-}hp(the\ ho)$
by(*induction rule*: *get-min.induct*)(*auto*)

lemma *get-min-min*: $[\![\ ho \neq None;\ invar\ ho;\ x \in\# mset\text{-}hp(the\ ho)\ ]\!] \Longrightarrow get\text{-}min\ ho \leq x$
by(*induction ho rule*: *get-min.induct*)(*auto simp*: *invar-def simp flip*: *sum-mset-sum-list*)

lemma *mset-link*: $mset\text{-}hp\ (link\ h1\ h2) = mset\text{-}hp\ h1 + mset\text{-}hp\ h2$
by(*induction h1 h2 rule*: *link.induct*)(*auto simp*: *add-ac*)

lemma *mset-merge*: $mset\text{-}heap\ (merge\ ho1\ ho2) = mset\text{-}heap\ ho1 + mset\text{-}heap\ ho2$
by (*induction ho1 ho2 rule*: *merge.induct*)
(*auto simp add*: *mset-heap-def mset-link ac-simps*)

lemma *mset-insert*: $mset\text{-}heap\ (insert\ a\ ho) = \{\#a\#\} + mset\text{-}heap\ ho$
by(*cases ho*) (*auto simp add*: *mset-link mset-heap-def insert-def*)

lemma *mset-pass*$_1$: $sum\text{-}list(map\ mset\text{-}hp\ (pass_1\ hs)) = sum\text{-}list(map\ mset\text{-}hp\ hs)$
by(*induction hs rule*: $pass_1$*.induct*)
(*auto simp*: *mset-link split*: *option.split*)

lemma *mset-pass*$_2$: $mset\text{-}heap\ (pass_2\ hs) = sum\text{-}list(map\ mset\text{-}hp\ hs)$
by(*induction hs rule*: *merge-pairs.induct*)
(*auto simp*: *mset-link mset-heap-def split*: *option.split*)

lemma *mset-del-min*: $ho \neq None \Longrightarrow$
$mset\text{-}heap\ (del\text{-}min\ ho) = mset\text{-}heap\ ho - \{\#get\text{-}min\ ho\#\}$
by(*induction ho rule*: *del-min.induct*)

(*auto simp*: *mset-heap-Some mset-pass*$_1$ *mset-pass*$_2$)

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

interpretation *pairing*: *Priority-Queue-Merge*
where *empty* = *None* **and** *is-empty* = $\lambda h.\ h$ = *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(*standard*, *goal-cases*)
  case *1* **show** *?case* **by**(*simp add*: *mset-heap-def*)
next
  case (*2 q*) **thus** *?case* **by**(*auto simp add*: *mset-heap-def split*: *option.split*)
next
  case *3* **show** *?case* **by**(*simp add*: *mset-insert mset-merge*)
next
  case *4* **thus** *?case* **by**(*simp add*: *mset-del-min mset-heap-empty*)
next
  case (*5 q*) **thus** *?case* **using** *get-min-in*[*of q*]
    by(*auto simp add*: *eq-Min-iff get-min-min mset-heap-empty mset-heap-Some*)
next
  case *6* **thus** *?case* **by** (*simp add*: *invar-def*)
next
  case *7* **thus** *?case* **by**(*rule invar-insert*)
next
  case *8* **thus** *?case* **by** (*simp add*: *invar-del-min*)
next
  case *9* **thus** *?case* **by** (*simp add*: *mset-merge*)
next
  case *10* **thus** *?case* **by** (*simp add*: *invar-merge*)
qed

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.