IsaSAT: Heuristics and Code Generation

Mathias Fleury, Jasmin Blanchette, Peter Lammich

July 17, 2023

Contents

theory *Map-Fun-Rel*
imports *More-Sepref.WB-More-Refinement*
begin

0.0.1 Refinement from function to lists

Throughout our formalization, we often use functions at the most abstract level, that we refine to lists assuming a known domain.

One thing to remark is that I have changed my mind on how to do things. Before we refined things directly and kept the domain implicit. Nowadays, I make the domain explicit – even if this means that we have to duplicate the information of the domain through all the components of our state.

Definition **definition** *map-fun-rel* :: ‹(*nat* × *'key*) *set* ⇒ (*'b* × *'a*) *set* ⇒ (*'b list* × (*'key* ⇒ *'a*)) *set*› **where**
map-fun-rel-def-internal:
‹*map-fun-rel D R* = {(*m*, *f*). ∀ (*i*, *j*)∈*D*. *i* < *length m* ∧ (*m* ! *i*, *f j*) ∈ *R*}›

lemma *map-fun-rel-def*:
‹⟨*R*⟩*map-fun-rel D* = {(*m*, *f*). ∀ (*i*, *j*)∈*D*. *i* < *length m* ∧ (*m* ! *i*, *f j*) ∈ *R*}›
unfolding *relAPP-def map-fun-rel-def-internal* **by** *auto*

lemma *map-fun-rel-nth*:
‹(*xs*,*ys*)∈⟨*R*⟩*map-fun-rel D* ⟹ (*i*,*j*)∈*D* ⟹ (*xs* ! *i* , *ys j*) ∈ *R*›
unfolding *map-fun-rel-def* **by** *auto*

In combination with lists **definition** *length-ll-f* **where**
‹*length-ll-f W L* = *length* (*W L*)›

lemma *map-fun-rel-length*:
‹(*xs*,*ys*)∈⟨⟨*R*⟩*list-rel*⟩*map-fun-rel D* ⟹ (*i*,*j*)∈*D* ⟹ (*length-ll xs i*, *length-ll-f ys j*) ∈ *nat-rel*›

unfolding *map-fun-rel-def* **by** (*auto simp*: *length-ll-def length-ll-f-def*
dest!: *bspec list-rel-imp-same-length*)

definition *append-update* :: ‹(′*a* ⇒ ′*b list*) ⇒ ′*a* ⇒ ′*b* ⇒ ′*a* ⇒ ′*b list*› **where**
‹*append-update W L a* = *W*(*L*:= *W* (*L*) @ [*a*])›

end

0.1 Pairing Heap According to Oksaki (Modified)

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

Chapter 1

Pairing heaps

To make it useful we simply parametrized the formalization by the order. We ruse the formalization of Tobias Nipkow, but make it *useful* for refinement by separating node and score. We also need to add way to increase the score.

1.0.1 Definitions

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

datatype (*'b*, *'a*) *hp* = *Hp* (*node*: *'b*) (*score*: *'a*) (*hps*: (*'b*, *'a*) *hp list*)

type-synonym (*'a*, *'b*) *heap* = (*'a*, *'b*) *hp option*

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

fun *get-min* :: (*'b*, *'a*) *heap* ⇒ *'a* **where**
get-min (*Some*(*Hp* - *x* -)) = *x*

This is basically the useful version:

fun *get-min2* :: (*'b*, *'a*) *heap* ⇒ *'b* **where**
get-min2 (*Some*(*Hp n x* -)) = *n*

locale *pairing-heap-assms* =
  fixes *lt* :: ‹*'a* ⇒ *'a* ⇒ *bool*› **and**
    le :: ‹*'a* ⇒ *'a* ⇒ *bool*›
begin

fun *link* :: (*'b*, *'a*) *hp* ⇒ (*'b*, *'a*) *hp* ⇒ (*'b*, *'a*) *hp* **where**
link (*Hp m x lx*) (*Hp n y ly*) =
  (*if lt x y then Hp m x* (*Hp n y ly* # *lx*) *else Hp n y* (*Hp m x lx* # *ly*))

fun *merge* :: (*'b*, *'a*) *heap* ⇒ (*'b*, *'a*) *heap* ⇒ (*'b*, *'a*) *heap* **where**
merge h None = *h* |
merge None h = *h* |
merge (*Some h1*) (*Some h2*) = *Some*(*link h1 h2*)

lemma *merge-None*[*simp*]: *merge None h* = *h*
by(*cases h*)*auto*

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

fun $pass_1$ :: ('b, 'a) *hp list* ⇒ ('b, 'a) *hp list* **where**
  $pass_1$ [] = []
| $pass_1$ [*h*] = [*h*]
| $pass_1$ (*h1*#*h2*#*hs*) = *link h1 h2* # $pass_1$ *hs*

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

fun (**in** −)*remove-key-children* :: ‹'b ⇒ ('b, 'a) *hp list* ⇒ ('b, 'a) *hp list*› **where**
  ‹*remove-key-children k* [] = []› |
  ‹*remove-key-children k* ((*Hp x n c*) # *xs*) =
  (*if k* = *x then remove-key-children k xs else* ((*Hp x n* (*remove-key-children k c*)) # *remove-key-children k xs*))›

fun (**in** −)*remove-key* :: ‹'b ⇒ ('b, 'a) *hp* ⇒ ('b, 'a) *heap*› **where**
  ‹*remove-key k* (*Hp x n c*) = (*if x* = *k then None else Some* (*Hp x n* (*remove-key-children k c*)))›

fun (**in** −)*find-key-children* :: ‹'b ⇒ ('b, 'a) *hp list* ⇒ ('b, 'a) *heap*› **where**
  ‹*find-key-children k* [] = *None*› |
  ‹*find-key-children k* ((*Hp x n c*) # *xs*) =
  (*if k* = *x then Some* (*Hp x n c*) *else*
  (*case find-key-children k c of Some a* ⇒ *Some a* | - ⇒ *find-key-children k xs*))›

fun (**in** −)*find-key* :: ‹'b ⇒ ('b, 'a) *hp* ⇒ ('b, 'a) *heap*› **where**
  ‹*find-key k* (*Hp x n c*) =
  (*if k* = *x then Some* (*Hp x n c*) *else find-key-children k c*)›

definition *decrease-key* :: ‹'b ⇒ 'a ⇒ ('b, 'a) *hp* ⇒ ('b, 'a) *heap*› **where**
  ‹*decrease-key k s hp* = (*case find-key k hp of None* ⇒ *Some hp*
    | (*Some* (*Hp* - - *c*)) ⇒
      (*case remove-key k hp of*
          None ⇒ *Some* (*Hp k s c*)
        | *Some x* ⇒ *merge-pairs* [*Hp k s c*, *x*]))›

1.0.2 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*]

Invariants

fun (**in** −) *set-hp* :: ‹(′*b*, ′*a*) *hp* ⇒ ′*a set*› **where**
‹*set-hp* (*Hp* - *x hs*) = ({*x*} ∪ ⋃(*set-hp* ‘ *set hs*))›

fun *php* :: (′*b*, ′*a*) *hp* ⇒ *bool* **where**
php (*Hp* - *x hs*) = (∀ *h* ∈ *set hs*. (∀ *y* ∈ *set-hp h*. *le x y*) ∧ *php h*)

definition *invar* :: (′*b*, ′*a*) *heap* ⇒ *bool* **where**
invar ho = (*case ho of None* ⇒ *True* | *Some h* ⇒ *php h*)
end

locale *pairing-heap* = *pairing-heap-assms lt le*
for *lt* :: ‹′*a* ⇒ ′*a* ⇒ *bool*› **and**
le :: ‹′*a* ⇒ ′*a* ⇒ *bool*› +
assumes *le*: ‹⋀*a b*. *le a b* ⟷ *a* = *b* ∨ *lt a b*› **and**
trans: ‹*transp le*› **and**
transt: ‹*transp lt*› **and**
totalt: ‹*totalp lt*›
begin

lemma *php-link*: *php h1* ⟹ *php h2* ⟹ *php* (*link h1 h2*)
apply (*induction h1 h2 rule*: *link.induct*)
apply (*auto 4 3 simp*: *le transt dest*: *transpD*[*OF transt*] *totalpD*[*OF totalt*])
by (*metis totalpD totalt transpD transt*)

lemma *invar-None*[*simp*]: ‹*invar None*›
by (*auto simp*: *invar-def*)

lemma *invar-merge*:
⟦ *invar h1*; *invar h2* ⟧ ⟹ *invar* (*merge h1 h2*)
by (*auto simp*: *php-link invar-def split*: *option.splits*)

lemma *invar-insert*: *invar h* ⟹ *invar* (*insert n x h*)
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 h* ⟹ *invar* (*del-min h*)
by(*induction h rule*: *del-min.induct*)
(*auto simp*: *invar-Some intro*!: *invar-pass1 invar-pass2*)

lemma (**in** −)*in-remove-key-children-in-childrenD*: ‹*h* ∈ *set* (*remove-key-children k c*) ⟹ *xa* ∈ *set-hp h* ⟹ *xa* ∈ ⋃(*set-hp* ‘ *set c*)›
by (*induction k c arbitrary*: *h rule*: *remove-key-children.induct*)

(*auto split*: *if-splits*)

lemma *php-remove-key-children*: ‹∀ *h*∈*set h1*. *php h* ⟹ *h* ∈ *set* (*remove-key-children k h1*) ⟹ *php h*›
  by (*induction k h1 arbitrary*: *h rule*: *remove-key-children.induct*; *simp*)
   (*force simp*: *decrease-key-def invar-def split*: *option.splits hp.splits if-splits*
    dest: *in-remove-key-children-in-childrenD*)

lemma *php-remove-key*: ‹*php h1* ⟹ *invar* (*remove-key k h1*)›
  by (*induction k h1 rule*: *remove-key.induct*)
   (*auto simp*: *decrease-key-def invar-def split*: *option.splits hp.splits*
    dest: *in-remove-key-children-in-childrenD*
    intro: *php-remove-key-children*)

lemma *invar-find-key-children*: ‹∀ *h*∈*set c*. *php h* ⟹ *invar* (*find-key-children k c*)›
  by (*induction k c rule*: *find-key-children.induct*)
   (*auto simp*: *invar-def split*: *option.splits*)

lemma *invar-find-key*: ‹*php h1* ⟹ *invar* (*find-key k h1*)›
  by (*induction k h1 rule*: *find-key.induct*)
   (*use invar-find-key-children*[*of* -] **in** ‹*force simp*: *invar-def*›)

lemma (**in** −)*remove-key-None-iff*: ‹*remove-key k h1* = *None* ⟷ *node h1* = *k*›
  by (*cases* ‹(*k*,*h1*)› *rule*: *remove-key.cases*) *auto*

lemma *php-decrease-key*:
  ‹*php h1* ⟹ (*case* (*find-key k h1*) *of None* ⇒ *True* | *Some a* ⇒ *le s* (*score a*)) ⟹ *invar* (*decrease-key k s h1*)›
  using *invar-find-key*[*of h1 k*, *simplified*] *remove-key-None-iff*[*of k h1*] *php-remove-key*[*of h1 k*]
  apply (*auto simp*: *decrease-key-def invar-def php-remove-key php-link*
    dest: *transpD*[*OF transt*, *of* - ‹*score* (*the* (*find-key k h1*))›] *totalpD*[*OF totalt*]
    split: *option.splits hp.splits*)
  apply (*meson le local.trans transpE*)
  apply (*rule php-link*)
  apply (*auto simp*: *decrease-key-def invar-def php-remove-key php-link*
    split: *option.splits hp.splits*
    dest: *transpD*[*OF transt*, *of* - ‹*score* (*the* (*find-key k h1*))›] *totalpD*[*OF totalt*])
  apply (*meson le local.trans transpE*)
  done

Functional Correctness

fun (**in** −) *mset-hp* :: (′*b*, ′*a*) *hp* ⇒′*a multiset* **where**
mset-hp (*Hp* - *x hs*) = {#*x*#} + *sum-mset*(*mset*(*map mset-hp hs*))

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

lemma (**in** −) *set-mset-mset-hp*: *set-mset* (*mset-hp h*) = *set-hp h*
by(*induction h*) *auto*

lemma (**in** −) *mset-hp-empty*[*simp*]: *mset-hp hp* ≠ {#}
by (*cases hp*) *auto*

lemma (**in** −) *mset-heap-Some*: *mset-heap*(*Some hp*) = *mset-hp hp*
by(*simp add*: *mset-heap-def*)

lemma (**in** −) *mset-heap-empty*: *mset-heap h* = {#} ⟷ *h* = *None*
by (*cases h*) (*auto simp add*: *mset-heap-def*)

lemma (**in** −)*get-min-in*:
h ≠ *None* ⟹ *get-min h* ∈ *set-hp*(*the h*)
by(*induction rule*: *get-min.induct*)(*auto*)

lemma *get-min-min*: ⟦ *h* ≠ *None*; *invar h*; *x* ∈ *set-hp*(*the h*) ⟧ ⟹ *le* (*get-min h*) *x*
by (*induction h rule*: *get-min.induct*) (*auto simp*: *invar-def le*)

lemma (**in** *pairing-heap-assms*) *mset-link*: *mset-hp* (*link h1 h2*) = *mset-hp h1* + *mset-hp h2*
by(*induction h1 h2 rule*: *link.induct*)(*auto simp*: *add-ac*)

lemma (**in** *pairing-heap-assms*) *mset-merge*: *mset-heap* (*merge h1 h2*) = *mset-heap h1* + *mset-heap h2*
by (*induction h1 h2 rule*: *merge.induct*)
(*auto simp add*: *mset-heap-def mset-link ac-simps*)

lemma (**in** *pairing-heap-assms*) *mset-insert*: *mset-heap* (*insert n a h*) = {#*a*#} + *mset-heap h*
by(*cases h*) (*auto simp add*: *mset-link mset-heap-def insert-def*)

lemma (**in** *pairing-heap-assms*) *mset-merge-pairs*: *mset-heap* (*merge-pairs hs*) = *sum-mset*(*image-mset mset-hp* (*mset hs*))
by(*induction hs rule*: *merge-pairs.induct*)
(*auto simp*: *mset-merge mset-link mset-heap-def Let-def split*: *option.split*)

lemma (**in** *pairing-heap-assms*) *mset-del-min*: *h* ≠ *None* ⟹
mset-heap (*del-min h*) = *mset-heap h* − {#*get-min h*#}
by(*induction h rule*: *del-min.induct*)
(*auto simp*: *mset-heap-Some pass12-merge-pairs mset-merge-pairs*)

Some more lemmas to make the heaps easier to use:

lemma *invar-merge-pairs*:
⟦∀ *h*∈*set h1*. *invar* (*Some h*)⟧ ⟹ *invar* (*merge-pairs h1*)
by (*metis invar-Some invar-pass1 invar-pass2 pass12-merge-pairs*)

end

context *pairing-heap-assms*
begin

lemma *merge-pairs-None-iff* [*iff*]: *merge-pairs hs* = *None* ⟷ *hs* = []
by (*cases hs rule*: *merge-pairs.cases*) *auto*

end

Last step: prove all axioms of the priority queue specification with the right sort:

locale *pairing-heap2* =
fixes *ltype* :: ‹′*a*::*linorder itself*›
begin

sublocale *pairing-heap* **where**
lt =‹(<) :: ′*a* ⇒ ′*a* ⇒ *bool*› **and** *le* = ‹(≤)›
by *unfold-locales*

(*auto simp*: *antisymp-def totalp-on-def*)

interpretation *pairing*: *Priority-Queue-Merge*
where *empty* = *None* **and** *is-empty* = $\lambda h.\ h = None$
and *merge* = *merge* **and** *insert* = ‹*insert default*›
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 set-mset-mset-hp*)
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

end
theory *Heaps-Abs*
  imports *Ordered-Pairing-Heap-List2*
    Weidenbach-Book-Base.Explorer
    Isabelle-LLVM.IICF
    More-Sepref.WB-More-Refinement
begin

We first tried to follow the setup from Isabelle LLVM, but it is not clear how useful this really is. Hence we adapted the definition from the abstract operations.

locale *hmstruct-with-prio* =
    fixes *lt* :: ‹$'v \Rightarrow 'v \Rightarrow$ *bool*› **and**
    le :: ‹$'v \Rightarrow 'v \Rightarrow$ *bool*›
  assumes *hm-le*: ‹$\bigwedge a\ b.\ le\ a\ b \longleftrightarrow a = b \lor lt\ a\ b$› **and**
    hm-trans: ‹*transp le*› **and**
    hm-transt: ‹*transp lt*› **and**
    hm-totalt: ‹*totalp lt*›
begin

    definition *prio-peek-min* **where**
      prio-peek-min ≡ $(\lambda(\mathcal{A},\ b,\ w).\ (\lambda v.$
        $v \in\#\ b$
      $\land\ (\forall\, v' \in set\text{-}mset\ b.\ le\ (w\ v)\ (w\ v'))))$

definition *mop-prio-peek-min* **where**
  $mop\text{-}prio\text{-}peek\text{-}min \equiv (\lambda(\mathcal{A},\ b,\ w).\ doN\ \{ASSERT\ (b{\neq}\{\#\});\ SPEC\ (prio\text{-}peek\text{-}min\ (\mathcal{A},\ b,w))\})$

definition *mop-prio-change-weight* **where**
  $mop\text{-}prio\text{-}change\text{-}weight \equiv (\lambda v\ \omega\ (\mathcal{A},\ b,\ w).\ doN\ \{$
    $ASSERT\ (v \in\#\ \mathcal{A});$
    $ASSERT\ (v \in\#\ b \longrightarrow le\ \omega\ (w\ v));$
    $RETURN\ (\mathcal{A},\ b,\ w(v := \omega))$
  $\})$

definition *mop-prio-insert* **where**
  $mop\text{-}prio\text{-}insert \equiv (\lambda v\ \omega\ (\mathcal{A},\ b,\ w).\ doN\ \{$
    $ASSERT\ (v \notin\#\ b \wedge\ v{\in}\#\mathcal{A});$
    $RETURN\ (\mathcal{A},\ add\text{-}mset\ v\ b,\ w(v := \omega))$
  $\})$

definition *mop-prio-is-in* **where**
  ‹$mop\text{-}prio\text{-}is\text{-}in = (\lambda v\ (\mathcal{A},\ b,\ w).\ do\ \{$
  $ASSERT\ (v \in\#\ \mathcal{A});$
  $RETURN\ (v \in\# b)$
  $\})$›
definition *mop-prio-insert-maybe* **where**
  $mop\text{-}prio\text{-}insert\text{-}maybe \equiv (\lambda v\ \omega\ (bw).\ doN\ \{$
    $b \leftarrow mop\text{-}prio\text{-}is\text{-}in\ v\ bw;$
    $if\ \neg b\ then\ mop\text{-}prio\text{-}insert\ v\ \omega\ (bw)$
    $else\ mop\text{-}prio\text{-}change\text{-}weight\ v\ \omega\ (bw)$
  $\})$

TODO this is a shortcut and it could make sense to force w to remember the old values.

definition *mop-prio-old-weight* **where**
  $mop\text{-}prio\text{-}old\text{-}weight = (\lambda v\ (\mathcal{A},\ b,\ w).\ doN\ \{$
    $ASSERT\ (v \in\#\ \mathcal{A});$
    $b \leftarrow mop\text{-}prio\text{-}is\text{-}in\ v\ (\mathcal{A},\ b,\ w);$
    $if\ b\ then\ RETURN\ (w\ v)\ else\ RES\ UNIV$
  $\})$

definition *mop-prio-insert-raw-unchanged* **where**
  $mop\text{-}prio\text{-}insert\text{-}raw\text{-}unchanged = (\lambda v\ h.\ doN\ \{$
    $ASSERT\ (v \notin\#\ fst\ (snd\ h));$
    $w \leftarrow mop\text{-}prio\text{-}old\text{-}weight\ v\ h;$
    $mop\text{-}prio\text{-}insert\ v\ w\ h$
  $\})$

definition *mop-prio-insert-unchanged* **where**
  $mop\text{-}prio\text{-}insert\text{-}unchanged = (\lambda v\ (bw).\ doN\ \{$
    $b \leftarrow mop\text{-}prio\text{-}is\text{-}in\ v\ bw;$
    $if\ \neg b\ then\ mop\text{-}prio\text{-}insert\text{-}raw\text{-}unchanged\ v\ (bw)$
    $else\ RETURN\ bw$
  $\})$

definition *prio-del* **where**
  ‹$prio\text{-}del = (\lambda v\ (\mathcal{A},\ b,\ w).\ (\mathcal{A},\ b - \{\#v\#\},\ w))$›

definition *mop-prio-del* **where**
  $mop\text{-}prio\text{-}del = (\lambda v\ (\mathcal{A},\ b,\ w).\ doN\ \{$
    $ASSERT\ (v \in\#\ b \wedge v \in\#\ \mathcal{A});$

RETURN (*prio-del v* ($\mathcal{A}$, *b*, *w*))
})

definition *mop-prio-pop-min* **where**
mop-prio-pop-min = ($\lambda\mathcal{A}bw$. *doN* {
v ← *mop-prio-peek-min* $\mathcal{A}bw$;
bw ← *mop-prio-del v* $\mathcal{A}bw$;
RETURN (*v*, *bw*)
})

sublocale *pairing-heap*
by *unfold-locales* (*rule hm-le hm-trans hm-transt hm-totalt*)+

end

end
theory *Pairing-Heaps*
imports *Ordered-Pairing-Heap-List2*
Isabelle-LLVM.IICF
More-Sepref.WB-More-Refinement
Heaps-Abs
begin

1.1 Pairing Heaps

1.1.1 Genealogy Over Pairing Heaps

We first tried to use the heapmap, but this attempt was a terrible failure, because as useful the heapmap is parametrized by the size. This might be useful in some contexts, but I consider this to be the most terrible idea ever, based on past experience. So instead I went for a modification of the pairing heaps.

To increase fun, we reuse the trick from VSIDS to represent the pairing heap inside an array in order to avoid allocation yet another array. As a side effect, it also avoids including the label inside the node (because per definition, the label is exactly the index). But maybe pointers are actually better, because by definition in Isabelle no graph is shared.

fun *mset-nodes* :: ('*b*, '*a*) *hp* ⇒'*b multiset* **where**
mset-nodes (*Hp x - hs*) = {#*x*#} + $\sum_{\#}$ (*mset-nodes* '# *mset hs*)

context *pairing-heap-assms*
begin

lemma *mset-nodes-link*[*simp*]: ‹*mset-nodes* (*link a b*) = *mset-nodes a* + *mset-nodes b*›
by (*cases a*; *cases b*)
auto

lemma *mset-nodes-merge-pairs*: ‹*merge-pairs a* ≠ *None* ⟹ *mset-nodes* (*the* (*merge-pairs a*)) = *sum-list* (*map mset-nodes a*)›
apply (*induction a rule*: *merge-pairs.induct*)
subgoal by *auto*
subgoal by *auto*
subgoal for *h1 h2 hs*
by (*cases hs*)
(*auto simp*: *Let-def split*: *option.splits*)
done

lemma *mset-nodes-pass*$_1$[*simp*]: ‹*sum-list* (*map mset-nodes* (*pass*$_1$ *a*)) = *sum-list* (*map mset-nodes a*)›
  apply (*induction a rule*: *pass*$_1$.*induct*)
  subgoal by *auto*
  subgoal by *auto*
  subgoal for *h1 h2 hs*
    by (*cases hs*)
     (*auto simp*: *Let-def split*: *option.splits*)
  done

lemma *mset-nodes-pass*$_2$[*simp*]: ‹*pass*$_2$ *a* ≠ *None* ⟹ *mset-nodes* (*the* (*pass*$_2$ *a*)) = *sum-list* (*map mset-nodes a*)›
  apply (*induction a rule*: *pass*$_1$.*induct*)
  subgoal by *auto*
  subgoal by *auto*
  subgoal for *h1 h2 hs*
    by (*cases hs*)
     (*auto simp*: *Let-def split*: *option.splits*)
  done

end

lemma *mset-nodes-simps*[*simp*]: ‹*mset-nodes* (*Hp x n hs*) = {#*x*#} + (*sum-list* (*map mset-nodes hs*))›
  by *auto*

lemmas [*simp del*] = *mset-nodes.simps*

fun *hp-next* **where**
  ‹*hp-next a* (*Hp m s* (*x* # *y* # *children*)) = (*if a* = *node x then Some y else* (*case hp-next a x of Some a* ⇒ *Some a* | *None* ⇒ *hp-next a* (*Hp m s* (*y* # *children*))))› |
  ‹*hp-next a* (*Hp m s* [*b*]) = *hp-next a b*› |
  ‹*hp-next a* (*Hp m s* []) = *None*›

lemma [*simp*]: ‹*size-list size* (*hps x*) < *Suc* (*size x* + *a*)›
  by (*cases x*) *auto*

fun *hp-prev* **where**
  ‹*hp-prev a* (*Hp m s* (*x* # *y* # *children*)) = (*if a* = *node y then Some x else* (*case hp-prev a x of Some a* ⇒ *Some a* | *None* ⇒ *hp-prev a* (*Hp m s* (*y* # *children*))))› |
  ‹*hp-prev a* (*Hp m s* [*b*]) = *hp-prev a b*› |
  ‹*hp-prev a* (*Hp m s* []) = *None*›

fun *hp-child* **where**
  ‹*hp-child a* (*Hp m s* (*x* # *children*)) = (*if a* = *m then Some x else* (*case hp-child a x of None* ⇒ *hp-child a* (*Hp m s children*) | *Some a* ⇒ *Some a*))› |
  ‹*hp-child a* (*Hp m s* -) = *None*›

fun *hp-node* **where**
  ‹*hp-node a* (*Hp m s* (*x*#*children*)) = (*if a* = *m then Some* (*Hp m s* (*x*#*children*)) *else* (*case hp-node a x of None* ⇒ *hp-node a* (*Hp m s children*) | *Some a* ⇒ *Some a*))› |
  ‹*hp-node a* (*Hp m s* []) = (*if a* = *m then Some* (*Hp m s* []) *else None*)›

lemma *node-in-mset-nodes*[*simp*]: ‹*node x* ∈# *mset-nodes x*›
  by (*cases x*; *auto*)

lemma *hp-next-None-notin*[*simp*]: ‹$m \notin\# $ *mset-nodes a* $\Longrightarrow$ *hp-next m a* = *None*›
by (*induction m a rule*: *hp-next.induct*) *auto*

lemma *hp-prev-None-notin*[*simp*]: ‹$m \notin\# $ *mset-nodes a* $\Longrightarrow$ *hp-prev m a* = *None*›
by (*induction m a rule*: *hp-prev.induct*) *auto*

lemma *hp-child-None-notin*[*simp*]: ‹$m \notin\# $ *mset-nodes a* $\Longrightarrow$ *hp-child m a* = *None*›
by (*induction m a rule*: *hp-child.induct*) *auto*

lemma *hp-node-None-notin2*[*iff*]: ‹*hp-node m a* = *None* $\longleftrightarrow$ $m \notin\# $ *mset-nodes a*›
by (*induction m a rule*: *hp-node.induct*) *auto*

lemma *hp-node-None-notin*[*simp*]: ‹$m \notin\# $ *mset-nodes a* $\Longrightarrow$ *hp-node m a* = *None*›
by *auto*

lemma *hp-node-simps*[*simp*]: ‹*hp-node m* (*Hp m* w_m ch_m) = *Some* (*Hp m* w_m ch_m)›
by (*cases* ch_m) *auto*

lemma *hp-next-None-notin-children*[*simp*]: ‹$a \notin\# $ *sum-list* (*map mset-nodes children*) $\Longrightarrow$
hp-next a (*Hp m* w_m (*children*)) = *None*›
by (*induction a* ‹*Hp m* w_m *children*› *arbitrary*:*children rule*: *hp-next.induct*) *auto*

lemma *hp-prev-None-notin-children*[*simp*]: ‹$a \notin\# $ *sum-list* (*map mset-nodes children*) $\Longrightarrow$
hp-prev a (*Hp m* w_m (*children*)) = *None*›
by (*induction a* ‹*Hp m* w_m *children*› *arbitrary*:*children rule*: *hp-prev.induct*) *auto*

lemma *hp-child-None-notin-children*[*simp*]: ‹$a \notin\# $ *sum-list* (*map mset-nodes children*) $\Longrightarrow$ $a \neq m$ $\Longrightarrow$
hp-child a (*Hp m* w_m (*children*)) = *None*›
by (*induction a* ‹*Hp m* w_m *children*› *arbitrary*:*children rule*: *hp-next.induct*) *auto*

The function above are nicer for definition than for usage. Instead we define the list version and change the simplification lemmas. We initially tried to use a recursive function, but the proofs did not go through (and it seemed that the induction principle were to weak).

fun *hp-next-children* **where**
‹*hp-next-children a* (*x* # *y* # *children*) = (*if a* = *node x then Some y else* (*case hp-next a x of Some a* ⇒ *Some a* | *None* ⇒ *hp-next-children a* (*y* # *children*)))› |
‹*hp-next-children a* [*b*] = *hp-next a b*› |
‹*hp-next-children a* [] = *None*›

lemma *hp-next-simps*[*simp*]:
‹*hp-next a* (*Hp m s children*) = *hp-next-children a children*›
by (*induction a children rule*: *hp-next-children.induct*) (*auto split*: *option.splits*)

lemma *hp-next-children-None-notin*[*simp*]: ‹$m \notin\# \sum_{\#}$ (*mset-nodes* '# *mset children*) $\Longrightarrow$ *hp-next-children m children* = *None*›
by (*induction m children rule*: *hp-next-children.induct*) *auto*

lemma [*simp*]: ‹*distinct-mset* (*mset-nodes a*) $\Longrightarrow$ *hp-next* (*node a*) *a* = *None*›
by (*induction a*) *auto*

lemma [*simp*]:
‹$ch_m \neq []$ $\Longrightarrow$ *hp-next-children* (*node a*) (*a* # ch_m) = *Some* (*hd* ch_m)›
by (*cases* ch_m) *auto*

fun *hp-prev-children* **where**

‹hp-prev-children a (x # y # children) = (if a = node y then Some x else (case hp-prev a x of Some a ⇒ Some a | None ⇒ hp-prev-children a (y # children)))› |
‹hp-prev-children a [b] = hp-prev a b› |
‹hp-prev-children a [] = None›

lemma hp-prev-simps[simp]:
‹hp-prev a (Hp m s children) = hp-prev-children a children›
by (induction a children rule: hp-prev-children.induct) (auto split: option.splits)

lemma hp-prev-children-None-notin[simp]: ‹m ∉# $\sum_{\#}$ (mset-nodes '# mset children) ⟹ hp-prev-children m children = None›
by (induction m children rule: hp-prev-children.induct) auto

lemma [simp]: ‹distinct-mset (mset-nodes a) ⟹ hp-prev (node a) a = None›
by (induction a) auto

lemma hp-next-in-first-child [simp]: ‹distinct-mset (sum-list (map mset-nodes ch_m) + (mset-nodes a)) ⟹
xa ∈# mset-nodes a ⟹ xa ≠ node a ⟹
hp-next-children xa (a # ch_m) = (hp-next xa a)›
by (cases ch_m) (auto split: option.splits dest!: multi-member-split)

lemma hp-next-skip-hd-children:
‹distinct-mset (sum-list (map mset-nodes ch_m) + (mset-nodes a)) ⟹ xa ∈# $\sum_{\#}$ (mset-nodes '# mset ch_m) ⟹
xa ≠ node a ⟹ hp-next-children xa (a # ch_m) = hp-next-children xa (ch_m)›
apply (cases ch_m)
apply (auto split: option.splits dest!: multi-member-split)
done

lemma hp-prev-in-first-child [simp]: ‹distinct-mset
(sum-list (map mset-nodes ch_m) + (mset-nodes a)) ⟹ xa ∈# mset-nodes a ⟹ hp-prev-children xa (a # ch_m) = hp-prev xa a›
by (cases ch_m) (auto split: option.splits dest!: multi-member-split)

lemma hp-prev-skip-hd-children:
‹distinct-mset (sum-list (map mset-nodes ch_m) + (mset-nodes a)) ⟹ xa ∈# $\sum_{\#}$ (mset-nodes '# mset ch_m) ⟹
xa ≠ node (hd ch_m) ⟹ hp-prev-children xa (a # ch_m) = hp-prev-children xa ch_m›
apply (cases ch_m)
apply (auto split: option.splits dest!: multi-member-split)
done

lemma node-hd-in-sum[simp]: ‹ch_m ≠ [] ⟹ node (hd ch_m) ∈# sum-list (map mset-nodes ch_m)›
by (cases ch_m) auto

lemma hp-prev-cadr-node[simp]: ‹ch_m ≠ [] ⟹ hp-prev-children (node (hd ch_m)) (a # ch_m) = Some a›
by (cases ch_m) auto

lemma hp-next-children-simps[simp]:
‹a = node x ⟹ hp-next-children a (x # y # children) = Some y›
‹a ≠ node x ⟹ hp-next a x ≠ None ⟹ hp-next-children a (x # children) = hp-next a x›
‹a ≠ node x ⟹ hp-next a x = None ⟹ hp-next-children a (x # children) = hp-next-children a (children)›
apply (solves auto)

apply (*solves* ‹*cases children*; *auto*›)+
done

lemma *hp-prev-children-simps*[*simp*]:
‹$a = node\ y \Longrightarrow hp\text{-}prev\text{-}children\ a\ (x\ \#\ y\ \#\ children) = Some\ x$›
‹$a \neq node\ y \Longrightarrow hp\text{-}prev\ a\ x \neq None \Longrightarrow hp\text{-}prev\text{-}children\ a\ (x\ \#\ y\ \#\ children) = hp\text{-}prev\ a\ x$›
‹$a \neq node\ y \Longrightarrow hp\text{-}prev\ a\ x = None \Longrightarrow hp\text{-}prev\text{-}children\ a\ (x\ \#\ y\ \#\ children) = hp\text{-}prev\text{-}children\ a\ (y\ \#\ children)$›
by *auto*

lemmas [*simp del*] = *hp-next-children.simps*(*1*) *hp-next.simps*(*1*) *hp-prev.simps*(*1*) *hp-prev-children.simps*(*1*)

lemma *hp-next-children-skip-first-append*[*simp*]:
‹$xa \notin\#\ \sum_{\#}\ (mset\text{-}nodes\ `\#\ mset\ ch) \Longrightarrow hp\text{-}next\text{-}children\ xa\ (ch\ @\ ch') = hp\text{-}next\text{-}children\ xa\ ch'$›
apply (*induction xa ch rule*: *hp-next-children.induct*)
subgoal
by (*auto simp*: *hp-next-children.simps*(*1*))
subgoal
by (*cases ch′*)
(*auto simp*: *hp-next-children.simps*(*1*))
subgoal by *auto*
done

lemma *hp-prev-children-skip-first-append*[*simp*]:
‹$xa \notin\#\ \sum_{\#}\ (mset\text{-}nodes\ `\#\ mset\ ch) \Longrightarrow xa \neq node\ m \Longrightarrow hp\text{-}prev\text{-}children\ xa\ (ch\ @\ m\ \#\ ch') = hp\text{-}prev\text{-}children\ xa\ (m\#ch')$›
apply (*induction xa ch rule*: *hp-prev-children.induct*)
subgoal
by (*auto simp*: *hp-prev-children.simps*(*1*))
subgoal
by (*auto simp*: *hp-prev-children.simps*(*1*))
subgoal by *auto*
done

lemma *hp-prev-children-skip-Cons*[*simp*]:
‹$xa \notin\#\ \sum_{\#}\ (mset\text{-}nodes\ `\#\ mset\ ch') \Longrightarrow xa \in\#\ mset\text{-}nodes\ m \Longrightarrow hp\text{-}prev\text{-}children\ xa\ (m\ \#\ ch') = hp\text{-}prev\ xa\ m$›
apply (*induction ch′*)
subgoal
by (*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits*)
subgoal
by (*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits*)
done

definition *hp-child-children* **where**
‹*hp-child-children a* = *option-hd o* (*List.map-filter* (*hp-child a*))›

lemma *hp-child-children-Cons-if*:
‹*hp-child-children a* (*x* # *y*) = (*if hp-child a x* = *None then hp-child-children a y else hp-child a x*)›
by (*auto simp*: *hp-child-children-def List.map-filter-def split*: *list.splits*)

lemma *hp-child-children-simps*[*simp*]:
‹*hp-child-children a* [] = *None*›
‹$hp\text{-}child\ a\ x = None \Longrightarrow hp\text{-}child\text{-}children\ a\ (x\ \#\ y) = hp\text{-}child\text{-}children\ a\ y$›
‹$hp\text{-}child\ a\ x \neq None \Longrightarrow hp\text{-}child\text{-}children\ a\ (x\ \#\ y) = hp\text{-}child\ a\ x$›
by (*auto simp*: *hp-child-children-def List.map-filter-def split*: *list.splits*)

lemma *hp-child-hp-children-simps2*[*simp*]:
‹$x \neq a \Longrightarrow$ *hp-child x* (*Hp a b child*) = *hp-child-children x child*›
by (*induction child*) (*auto split*: *option.splits*)

lemma *hp-child-children-None-notin*[*simp*]: ‹$m \notin\#$ $\sum_{\#}$ (*mset-nodes* ‘# *mset children*) $\Longrightarrow$ *hp-child-children m children* = *None*›
by (*induction children*) *auto*

definition *hp-node-children* **where**
‹*hp-node-children a* = *option-hd o* (*List.map-filter* (*hp-node a*))›

lemma *hp-node-children-Cons-if*:
‹*hp-node-children a* (*x* # *y*) = (*if hp-node a x* = *None then hp-node-children a y else hp-node a x*)›
by (*auto simp*: *hp-node-children-def List.map-filter-def split*: *list.splits*)

lemma *hp-node-children-simps*[*simp*]:
‹*hp-node-children a* [] = *None*›
‹*hp-node a x* =*None* $\Longrightarrow$ *hp-node-children a* (*x* # *y*) = *hp-node-children a y*›
‹*hp-node a x* $\neq$ *None* $\Longrightarrow$ *hp-node-children a* (*x* # *y*) = *hp-node a x*›
by (*auto simp*: *hp-node-children-def List.map-filter-def split*: *list.splits*)

lemma *hp-node-children-simps2*[*simp*]:
‹$x \neq a \Longrightarrow$ *hp-node x* (*Hp a b child*) = *hp-node-children x child*›
by (*induction child*) (*auto split*: *option.splits*)

lemma *hp-node-children-None-notin2*: ‹*hp-node-children m children* = *None* $\longleftrightarrow$ $m \notin\#$ $\sum_{\#}$ (*mset-nodes* ‘# *mset children*)›
apply (*induction children*)
apply *auto*
by (*metis hp-node-children-simps*(*2*) *hp-node-children-simps*(*3*) *option-last-Nil option-last-Some-iff*(*2*))

lemma *hp-node-children-None-notin*[*simp*]: ‹$m \notin\#$ $\sum_{\#}$ (*mset-nodes* ‘# *mset children*) $\Longrightarrow$ *hp-node-children m children* = *None*›
by (*induction children*) *auto*

lemma *hp-next-children-hd-simps*[*simp*]:
‹*a* = *node x* $\Longrightarrow$ *distinct-mset* (*sum-list* (*map mset-nodes* (*x* # *children*))) $\Longrightarrow$
hp-next-children a (*x* # *children*) = *option-hd children*›
by (*cases children*) *auto*

lemma *hp-next-children-simps-if*:
‹ *distinct-mset* (*sum-list* (*map mset-nodes* (*x* # *children*))) $\Longrightarrow$
hp-next-children a (*x* # *children*) = (*if a* = *node x then option-hd children else case hp-next a x of None* ⇒ *hp-next-children a children* | *a* ⇒ *a*)›
by (*cases children*) (*auto split*: *if-splits option.splits*)

lemma *hp-next-children-skip-end*[*simp*]:
‹$n \in\#$ *mset-nodes a* $\Longrightarrow$ $n \neq$ *node a* $\Longrightarrow$ $n \notin\#$ *sum-list* (*map mset-nodes b*) $\Longrightarrow$
distinct-mset (*mset-nodes a*) $\Longrightarrow$
hp-next-children n (*a* # *b*) = *hp-next n a*›
by (*induction b*) (*auto simp add*: *hp-next-children.simps*(*1*) *split*: *option.splits*)

lemma *hp-next-children-append2*[*simp*]:

‹$x \neq n \Longrightarrow x \notin\# $ *sum-list* (*map mset-nodes* ch_m) $\Longrightarrow$ *hp-next-children* x (*Hp* n w_n ch_n # ch_m) = *hp-next-children* x ch_n›
by (*cases* ch_m) (*auto simp*: *hp-next-children.simps*(*1*) *split*: *option.splits*)

lemma *hp-next-children-skip-Cons-append*[*simp*]:
‹*NO-MATCH* [] *b* $\Longrightarrow$ *x* $\in\#$ *sum-list* (*map mset-nodes a*) $\Longrightarrow$
distinct-mset (*sum-list* (*map mset-nodes* (*a* @ *m* # *b*))) $\Longrightarrow$
hp-next-children x (*a* @ *m* # *b*) = *hp-next-children x* (*a* @ *m* # [])›
apply (*induction x a rule*: *hp-next-children.induct*)
apply (*auto simp*: *hp-next-children.simps*(*1*) *distinct-mset-add split*: *option.splits*)
apply (*metis* (*no-types*, *lifting*) *add-mset-disjoint*(*1*) *hp-next-children.simps*(*2*)
hp-next-children-None-notin hp-next-children-simps(*2*) *hp-next-children-simps*(*3*)
hp-next-children-skip-first-append mset-add node-in-mset-nodes sum-image-mset-sum-map union-iff)
by (*metis add-mset-disjoint*(*1*) *hp-next-None-notin hp-next-children-None-notin*
hp-next-children-simps(*3*) *insert-DiffM node-in-mset-nodes sum-image-mset-sum-map union-iff*)

lemma *hp-next-children-append-single-remove-children*:
‹*NO-MATCH* [] ch_m $\Longrightarrow$ *x* $\in\#$ *sum-list* (*map mset-nodes a*) $\Longrightarrow$
distinct-mset (*sum-list* (*map mset-nodes* (*a* @ [*Hp m* w_m ch_m]))) $\Longrightarrow$
map-option node (*hp-next-children x* (*a* @ [*Hp m* w_m ch_m])) =
map-option node (*hp-next-children x* (*a* @ [*Hp m* w_m []]))›
apply (*induction x a rule*: *hp-next-children.induct*)
apply (*auto simp*: *hp-next-children.simps*(*1*) *distinct-mset-add split*: *option.splits*)
apply (*smt* (*verit*, *ccfv-threshold*) *distinct-mset-add hp-next-None-notin hp-next-children.simps*(*2*)
hp-next-children-simps(*3*) *hp-next-children-skip-first-append hp-next-in-first-child hp-next-simps*
node-in-mset-nodes sum-image-mset-sum-map union-assoc union-commute)
apply (*simp add*: *disjunct-not-in*)
done

lemma *hp-prev-children-first-child*[*simp*]:
‹$m \neq n \Longrightarrow n \notin\#$ *sum-list* (*map mset-nodes b*) $\Longrightarrow$ $n \notin\#$ *sum-list* (*map mset-nodes* ch_n) $\Longrightarrow$
$n \in\#$ *sum-list* (*map mset-nodes child*) $\Longrightarrow$
hp-prev-children n (*Hp m* w_m *child* # *b*) = *hp-prev-children n child*›
by (*cases b*) (*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits*)

lemma *hp-prev-children-skip-last-append*[*simp*]:
‹*NO-MATCH* [] *ch'* $\Longrightarrow$
distinct-mset (*sum-list* (*map mset-nodes* (*ch* @*ch'*))) $\Longrightarrow$
xa $\notin\#$ $\sum_\#$ (*mset-nodes* '# *mset ch'*) $\Longrightarrow$ *xa* $\in\#$ $\sum_\#$ (*mset-nodes* '# *mset* (*ch*)) $\Longrightarrow$ *hp-prev-children xa* (*ch* @ *ch'*) = *hp-prev-children xa* (*ch*)›
apply (*induction xa ch rule*: *hp-prev-children.induct*)
subgoal for *a x y children*
by (*subgoal-tac* ‹*distinct-mset* (*sum-list* (*map mset-nodes* ((*y* # *children*) @ *ch'*)))›)
(*auto simp*: *hp-prev-children.simps*(*1*) *dest*!: *multi-member-split split*: *option.splits*
dest: *WB-List-More.distinct-mset-union2*)
subgoal
by (*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits dest*: *multi-member-split*)
subgoal by *auto*
done

lemma *hp-prev-children-Cons-append-found*[*simp*]:
‹$m \notin\#$ *sum-list* (*map mset-nodes a*) $\Longrightarrow$ $m \notin\#$ *sum-list* (*map mset-nodes ch*) $\Longrightarrow$ $m \notin\#$ *sum-list* (*map mset-nodes b*) $\Longrightarrow$ *hp-prev-children m* (*a* @ *Hp m* w_m *ch* # *b*) = *option-last a*›
by (*induction m a rule*: *hp-prev-children.induct*)
(*auto simp*: *hp-prev-children.simps*(*1*))

lemma *hp-prev-children-append-single-remove-children*:
‹*NO-MATCH* [] ch_m ⟹ x ∈# *sum-list* (*map mset-nodes a*) ⟹
distinct-mset (*sum-list* (*map mset-nodes* (*Hp m* w_m ch_m # *a*))) ⟹
map-option node (*hp-prev-children x* (*Hp m* w_m ch_m # *a*)) =
map-option node (*hp-prev-children x* (*Hp m* w_m [] # *a*))›
by (*induction a*) (*auto simp*: *hp-prev-children.simps*(*1*) *distinct-mset-add split*: *option.splits*
dest!: *multi-member-split*)

lemma *map-option-skip-in-child*:
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes a*))) ⟹ *m* ∉# *sum-list* (*map mset-nodes* ch_m) ⟹
ch_m ≠ [] ⟹
hp-prev-children (*node* (*hd* ch_m)) (*a* @ [*Hp m* w_m (*Hp n* w_n ch_n # ch_m)]) = *Some* (*Hp n* w_n ch_n)›
apply (*induction* ‹*node* (*hd* ch_m)› *a rule*: *hp-prev-children.induct*)
subgoal for *x y children*
by (*cases x*; *cases y*)
(*auto simp add*: *hp-prev-children.simps*(*1*) *disjunct-not-in distinct-mset-add*
split: *option.splits*)
subgoal for *b*
by (*cases b*)
(*auto simp*: *hp-prev-children.simps*(*1*) *disjunct-not-in distinct-mset-add*
split: *option.splits*)
subgoal by *auto*
done

lemma *hp-child-children-skip-first*[*simp*]:
‹*x* ∈# *sum-list* (*map mset-nodes ch′*) ⟹
distinct-mset (*sum-list* (*map mset-nodes ch*) + *sum-list* (*map mset-nodes ch′*)) ⟹
hp-child-children x (*ch* @ *ch′*) = *hp-child-children x ch′*›
apply (*induction ch*)
apply (*auto simp*: *hp-child-children-Cons-if dest*!: *multi-member-split*)
by (*metis WB-List-More.distinct-mset-union2 union-ac*(*1*))

lemma *hp-child-children-skip-last*[*simp*]:
‹*x* ∈# *sum-list* (*map mset-nodes ch*) ⟹
distinct-mset (*sum-list* (*map mset-nodes ch*) + *sum-list* (*map mset-nodes ch′*)) ⟹
hp-child-children x (*ch* @ *ch′*) = *hp-child-children x ch*›
apply (*induction ch*)
apply (*auto simp*: *hp-child-children-Cons-if dest*!: *multi-member-split*)
by (*metis WB-List-More.distinct-mset-union2 union-ac*(*1*))

lemma *hp-child-children-skip-last-in-first*:
‹*distinct-mset* (*sum-list* (*map mset-nodes* (*Hp m* w_m (*Hp n* w_n ch_n # ch_m) # *b*))) ⟹
hp-child-children n (*Hp m* w_m (*Hp n* w_n ch_n # ch_m) # *b*) = *hp-child n* (*Hp m* w_m (*Hp n* w_n ch_n # ch_m))›
by (*auto simp*: *hp-child-children-Cons-if split*: *option.splits*)

lemma *hp-child-children-hp-child*[*simp*]: ‹*hp-child-children x* [*a*] = *hp-child x a*›
by (*auto simp*: *hp-child-children-def List.map-filter-def*)

lemma *hp-next-children-last*[*simp*]:
‹*distinct-mset* (*sum-list* (*map mset-nodes a*)) ⟹ *a* ≠ [] ⟹
hp-next-children (*node* (*last a*)) (*a* @ *b*) = *option-hd b*›

apply (*induction* ‹*node* (*last a*)› *a rule*: *hp-next-children.induct*)
apply (*auto simp*: *hp-next-children.simps*(*1*) *dest*: *multi-member-split*)
apply (*metis add-diff-cancel-right′ distinct-mset-in-diff node-in-mset-nodes*)
apply (*metis add-diff-cancel-right′ distinct-mset-in-diff node-in-mset-nodes*)
apply (*metis Duplicate-Free-Multiset.distinct-mset-union2 add-diff-cancel-right′ distinct-mset-in-diff empty-append-eq-id hp-next-None-notin node-in-mset-nodes option.simps*(*4*))
apply (*metis Misc.last-in-set add-diff-cancel-left′ distinct-mem-diff-mset node-in-mset-nodes sum-list-map-remove1 union-iff*)
apply (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ append-butlast-last-id distinct-mem-diff-mset distinct-mset-add inter-mset-empty-distrib-right list.distinct*(*2*) *list.sel*(*1*) *map-append node-hd-in-sum node-in-mset-nodes sum-list.append*)
apply (*metis add-diff-cancel-right′ append-butlast-last-id distinct-mset-add distinct-mset-in-diff hp-next-None-notin list.sel*(*1*) *map-append node-hd-in-sum not-Cons-self2 option.case*(*1*) *sum-list.append union-iff*)
by (*metis* (*no-types*, *lifting*) *arith-simps*(*50*) *hp-next-children-hd-simps hp-next-children-simps*(*1*) *list.exhaust list.sel*(*1*) *list.simps*(*8*) *list.simps*(*9*) *option-hd-Some-iff*(*1*) *sum-list.Cons sum-list.Nil*)

lemma *hp-next-children-skip-last-not-last*:
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + *sum-list* (*map mset-nodes b*)) $\Longrightarrow$
a ≠ [] $\Longrightarrow$
x ≠ *node* (*last a*) $\Longrightarrow$ *x* ∈# *sum-list* (*map mset-nodes a*) $\Longrightarrow$
hp-next-children x (*a* @ *b*) = *hp-next-children x a*›
apply (*cases a rule*: *rev-cases*)
subgoal by *auto*
subgoal for *ys y*
apply (*cases* ‹*x* ∉# *mset-nodes* (*last a*)›)
subgoal by (*auto simp*: *ac-simps*)
subgoal
apply *auto*
apply (*subst hp-next-children-skip-first-append*)
apply (*auto simp*: *ac-simps*)
using *distinct-mset-in-diff* **apply** *fastforce*
using *distinct-mset-in-diff distinct-mset-union* **by** *fastforce*
done
done

lemma *hp-node-children-append-case*:
‹*hp-node-children x* (*a* @ *b*) = (*case hp-node-children x a of None* ⇒ *hp-node-children x b* | *x* ⇒ *x*)›
by (*auto simp*: *hp-node-children-def List.map-filter-def split*: *option.splits*)

lemma *hp-node-children-append*[*simp*]:
‹*hp-node-children x a* = *None* $\Longrightarrow$ *hp-node-children x* (*a* @ *b*) = *hp-node-children x b*›
‹*hp-node-children x a* ≠ *None* $\Longrightarrow$ *hp-node-children x* (*a* @ *b*) = *hp-node-children x a*›
by (*auto simp*: *hp-node-children-append-case*)

lemma *ex-hp-node-children-Some-in-mset-nodes*:
‹(∃ *y*. *hp-node-children xa a* = *Some y*) ⟷ *xa* ∈# *sum-list* (*map mset-nodes a*)›
using *hp-node-children-None-notin2*[*of xa a*] **by** *auto*

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN NEMonad.SPEC*

lemma *hp-node-node-itself*[*simp*]: ‹*hp-node* (*node x2*) *x2* = *Some x2*›
by (*cases x2*; *cases* ‹*hps x2*›) *auto*

lemma *hp-child-hd*[*simp*]: ‹*hp-child x1* (*Hp x1 x2 x3*) = *option-hd x3*›

by (*cases x3*) *auto*

lemma *drop-is-single-iff*: ‹*drop e xs* = [*a*] ⟷ *last xs* = *a* ∧ *e* = *length xs* − *1* ∧ *xs* ≠ []›
apply *auto*
apply (*metis append-take-drop-id last-snoc*)
by (*metis diff-diff-cancel diff-is-0-eq′ length-drop length-list-Suc-0 n-not-Suc-n nat-le-linear*)

lemma *distinct-mset-mono′*: ‹*distinct-mset D* ⟹ *D′* ⊆# *D* ⟹ *distinct-mset D′*›
by (*metis distinct-mset-union subset-mset.le-iff-add*)

context *pairing-heap-assms*
begin

lemma $pass_1$*-append-even*: ‹*even* (*length xs*) ⟹ $pass_1$ (*xs* @ *ys*) = $pass_1$ *xs* @ $pass_1$ *ys*›
by (*induction xs rule*: $pass_1$*.induct*) *auto*

lemma $pass_2$*-None-iff*[*simp*]: ‹$pass_2$ *list* = *None* ⟷ *list* = []›
by (*cases list*)
auto

lemma *last-*$pass_1$[*simp*]: *odd* (*length xs*) ⟹ *last* ($pass_1$ *xs*) = *last xs*
by (*metis* $pass_1$*.simps*(*2*) *append-butlast-last-id even-Suc last-snoc length-append-singleton length-greater-0-conv odd-pos* $pass_1$*-append-even*)
end

lemma *get-min2-alt-def*: ‹*get-min2* (*Some h*) = *node h*›
by (*cases h*) *auto*

fun *hp-parent* :: ‹′*a* ⇒ (′*a*, ′*b*) *hp* ⇒ (′*a*, ′*b*)*hp option*› **where**
‹*hp-parent n* (*Hp a sc* (*x* # *children*)) = (*if n* = *node x then Some* (*Hp a sc* (*x* # *children*)) *else map-option the* (*option-hd* (*filter* ((≠) *None*) (*map* (*hp-parent n*) (*x*#*children*)))))› |
‹*hp-parent n* - = *None*›

definition *hp-parent-children* :: ‹′*a* ⇒ (′*a*, ′*b*) *hp list* ⇒ (′*a*, ′*b*)*hp option*› **where**
‹*hp-parent-children n xs* = *map-option the* (*option-hd* (*filter* ((≠) *None*) (*map* (*hp-parent n*) *xs*)))›

lemma *hp-parent-None-notin*[*simp*]: ‹*m* ∉# *mset-nodes a* ⟹ *hp-parent m a* = *None*›
apply (*induction m a rule*: *hp-parent.induct*)
apply (*auto simp*: *filter-empty-conv*)
by (*metis node-in-mset-nodes sum-list-map-remove1 union-iff*)

lemma *hp-parent-children-None-notin*[*simp*]: ‹(*m*) ∉# *sum-list* (*map mset-nodes a*) ⟹ *hp-parent-children m a* = *None*›
by (*induction a*)
(*auto simp*: *filter-empty-conv hp-parent-children-def*)

lemma *hp-parent-children-cons*: ‹*hp-parent-children a* (*x* # *children*) = (*case hp-parent a x of None* ⇒ *hp-parent-children a children* | *Some a* ⇒ *Some a*)›
by (*auto simp*: *hp-parent-children-def*)

lemma *hp-parent-simps-if*:
‹*hp-parent n* (*Hp a sc* (*x* # *children*)) = (*if n* = *node x then Some* (*Hp a sc* (*x* # *children*)) *else hp-parent-children n* (*x*#*children*))›

by (*auto simp*: *hp-parent-children-def*)

lemmas [*simp del*] = *hp-parent.simps*(*1*)

lemma *hp-parent-simps*:
‹*n* = *node x* ⟹ *hp-parent n* (*Hp a sc* (*x* # *children*)) = *Some* (*Hp a sc* (*x* # *children*))›
‹*n* ≠ *node x* ⟹ *hp-parent n* (*Hp a sc* (*x* # *children*)) = *hp-parent-children n* (*x* # *children*)›
by (*auto simp*: *hp-parent-simps-if*)

lemma *hp-parent-itself*[*simp*]: ‹*distinct-mset* (*mset-nodes x*) ⟹ *hp-parent* (*node x*) *x* = *None*›
by (*cases* ‹(*node x*, *x*)› *rule*: *hp-parent.cases*)
(*auto simp*: *hp-parent.simps hp-parent-children-def filter-empty-conv sum-list-map-remove1*)

lemma *hp-parent-children-itself*[*simp*]:
‹*distinct-mset* (*mset-nodes x* + *sum-list* (*map mset-nodes children*)) ⟹ *hp-parent-children* (*node x*) (*x* # *children*) = *None*›
by (*auto simp*: *hp-parent-children-def filter-empty-conv disjunct-not-in distinct-mset-add sum-list-map-remove1 dest*: *distinct-mset-union*)

lemma *hp-parent-in-nodes*: ‹*hp-parent n x* ≠ *None* ⟹ *node* (*the* (*hp-parent n x*)) ∈# *mset-nodes x*›
apply (*induction n x rule*: *hp-parent.induct*)
subgoal premises *p* **for** *n a sc x children*
using *p p*(*1*)[*of xa*]
apply (*auto simp*: *hp-parent.simps*)
apply (*cases* ‹*filter* (λ*y*. ∃ *ya*. *y* = *Some ya*) (*map* (*hp-parent n*) *children*)›)
apply (*fastforce simp*: *filter-empty-conv filter-eq-Cons-iff map-eq-append-conv*)+
done
subgoal for *n v va*
by (*auto simp*: *hp-parent.simps filter-empty-conv*)
done

lemma *hp-parent-children-Some-iff*:
‹*hp-parent-children a xs* = *Some y* ⟷ (∃ *u b as*. *xs* = *u* @ *b* # *as* ∧ (∀ *x*∈*set u*. *hp-parent a x* = *None*) ∧ *hp-parent a b* = *Some y*)›
by (*cases* ‹(*filter* (λ*y*. ∃ *ya*. *y* = *Some ya*) (*map* (*hp-parent a*) *xs*))›)
(*fastforce simp*: *hp-parent-children-def filter-empty-conv filter-eq-Cons-iff map-eq-append-conv*)+

lemma *hp-parent-children-in-nodes*:
‹*hp-parent-children b xs* ≠ *None* ⟹ *node* (*the* (*hp-parent-children b xs*)) ∈# ∑# (*mset-nodes* '# *mset xs*)›
by (*metis hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-append*(*1*)
hp-node-children-append(*2*) *hp-node-children-simps*(*3*) *hp-parent-children-Some-iff*
hp-parent-in-nodes option.collapse)

lemma *hp-parent-hp-child*:
‹*distinct-mset* ((*mset-nodes* (*a*::('*a*,*nat*)*hp*))) ⟹ *hp-child n a* ≠ *None* ⟹ *map-option node* (*hp-parent* (*node* (*the* (*hp-child n a*))) *a*) = *Some n*›
apply (*induction n a rule*: *hp-child.induct*)
subgoal for *n a sc x children*
apply (*simp add*: *hp-parent-simps-if*)
apply *auto*
subgoal for *y*
apply (*auto simp add*: *hp-parent-simps-if hp-parent-children-Some-iff*
split: *option.splits dest*: *distinct-mset-union*)
apply (*metis* (*no-types*, *lifting*) *diff-single-trivial disjunct-not-in distinct-mem-diff-mset*
distinct-mset-add hp-parent-None-notin mset-cancel-union(*2*) *mset-nodes-simps node-in-mset-nodes*

option-last-Nil option-last-Some-iff (*2*) *sum-mset-sum-list*)
done
subgoal for *y*
using *distinct-mset-union*[*of* ‹*mset-nodes x*› ‹*sum-list* (*map mset-nodes children*)›]
distinct-mset-union[*of* ‹*sum-list* (*map mset-nodes children*)› ‹*mset-nodes x*›]
apply (*auto simp add*: *hp-parent-simps-if ac-simps hp-parent-children-cons*
split: *option.splits dest*: *distinct-mset-union*)
apply (*metis Groups.add-ac*(*2*) *add-mset-add-single disjunct-not-in distinct-mset-add hp-parent-None-notin member-add-mset mset-nodes-simps*
option-last-Nil option-last-Some-iff (*2*) *sum-mset-sum-list*)
by (*metis hp.sel*(*1*) *hp-parent.simps*(*2*) *hp-parent-simps-if option.sel option.simps*(*3*) *pairing-heap-assms.pass*$_2$*.cases*)
done
subgoal by *auto*
done

lemma *hp-child-hp-parent*:
‹*distinct-mset* ((*mset-nodes* (*a*::('*a*,*nat*)*hp*))) ⟹ *hp-parent n a* ≠ *None* ⟹ *map-option node* (*hp-child* (*node* (*the* (*hp-parent n a*))) *a*) = *Some n*›
apply (*induction n a rule*: *hp-parent.induct*)
subgoal for *n a sc x children*
apply (*simp add*: *hp-parent-simps-if*)
apply *auto*
subgoal for *y*
using *distinct-mset-union*[*of* ‹*mset-nodes x*› ‹*sum-list* (*map mset-nodes children*)›]
distinct-mset-union[*of* ‹*sum-list* (*map mset-nodes children*)› ‹*mset-nodes x*›]
apply (*auto simp add*: *hp-parent-simps-if hp-parent-children-cons ac-simps*
split: *option.splits*)
apply (*smt* (*verit*, *del-insts*) *hp-parent-children-Some-iff hp-parent-in-nodes list.map*(*2*) *map-append option.sel option-last-Nil option-last-Some-iff* (*2*) *sum-list.append sum-list-simps*(*2*) *union-iff*)
by *fastforce*
subgoal premises *p* **for** *yy*
using *p*(*2*−) *p*(*1*)[*of x*]
using *distinct-mset-union*[*of* ‹*mset-nodes x*› ‹*sum-list* (*map mset-nodes children*)›]
distinct-mset-union[*of* ‹*sum-list* (*map mset-nodes children*)› ‹*mset-nodes x*›]
apply (*auto simp add*: *hp-parent-simps-if hp-parent-children-cons ac-simps*
split: *option.splits*)
apply (*smt* (*verit*, *del-insts*) *disjunct-not-in distinct-mset-add hp-child-None-notin*
hp-parent-children-Some-iff hp-parent-in-nodes list.map(*2*) *map-append option.sel*
option-last-Nil option-last-Some-iff (*2*) *sum-list.Cons sum-list.append union-iff*)

using *p*(*1*)
apply (*auto simp*: *hp-parent-children-Some-iff*)
by (*metis WB-List-More.distinct-mset-union2 distinct-mset-union hp-child-children-simps*(*3*)
hp-child-children-skip-first hp-child-hp-children-simps2 hp-parent-in-nodes list.map(*2*)
option.sel option-last-Nil option-last-Some-iff (*2*) *sum-list.Cons union-iff*)
done
subgoal by *auto*
done

lemma *hp-parent-children-append-case*:
‹*hp-parent-children a* (*xs* @ *ys*) = (*case hp-parent-children a xs of None* ⇒ *hp-parent-children a ys* | *Some a* ⇒ *Some a*)›
by (*auto simp*: *hp-parent-children-def comp-def option-hd-def*)

lemma *hp-parent-children-append-skip-first*[*simp*]:

‹$a \notin\# \sum_{\#}$ (*mset-nodes* ‘# *mset xs*) ⟹ *hp-parent-children a* (*xs* @ *ys*) = *hp-parent-children a ys*›
by (*auto simp*: *hp-parent-children-append-case split*: *option.splits*)

lemma *hp-parent-children-append-skip-second*[*simp*]:
‹$a \notin\# \sum_{\#}$ (*mset-nodes* ‘# *mset ys*) ⟹ *hp-parent-children a* (*xs* @ *ys*) = *hp-parent-children a xs*›
by (*auto simp*: *hp-parent-children-append-case split*: *option.splits*)

lemma *hp-parent-simps-single-if*:
‹*hp-parent n* (*Hp a sc* (*children*)) =
(*if children* = [] *then None else if n* = *node* (*hd children*) *then Some* (*Hp a sc* (*children*))
else hp-parent-children n children)›
by (*cases children*)
(*auto simp*: *hp-parent-simps*)

lemma *hp-parent-children-remove-key-children*:
‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *mset xs*)) ⟹ *hp-parent-children a* (*remove-key-children a xs*) = *None*›
apply (*induction a xs rule*: *remove-key-children.induct*)
subgoal by *auto*
subgoal for *k x n c xs*
apply (*auto simp*: *hp-parent-simps-if hp-parent-children-cons*
split: *option.split*
dest: *WB-List-More.distinct-mset-union2*)
apply (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims disjunct-not-in*
distinct-mset-add hp.sel(*1*) *hp-parent-simps-single-if list.map*(*2*) *list.sel*(*1*) *list.simps*(*3*)
node-hd-in-sum node-in-mset-nodes option-last-Nil option-last-Some-iff(*2*) *sum-list-simps*(*2*))
apply (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims disjunct-not-in*
distinct-mset-add hp.sel(*1*) *hp-parent-simps-single-if list.map*(*2*) *list.sel*(*1*) *list.simps*(*3*)
node-hd-in-sum node-in-mset-nodes option-last-Nil option-last-Some-iff(*2*) *sum-list-simps*(*2*))
done
done

lemma *remove-key-children-notin-unchanged*[*simp*]: ‹$x \notin\#$ *sum-list* (*map mset-nodes c*) ⟹ *remove-key-children x c* = *c*›
by (*induction x c rule*: *remove-key-children.induct*)
auto

lemma *remove-key-notin-unchanged*[*simp*]: ‹$x \notin\#$ *mset-nodes c* ⟹ *remove-key x c* = *Some c*›
by (*induction x c rule*: *remove-key.induct*)
auto

lemma *remove-key-remove-all*: ‹$k \notin\# \sum_{\#}$ (*mset-nodes* ‘# *mset* (*remove-key-children k c*))›
by (*induction k c rule*: *remove-key-children.induct*) *auto*

lemma *hd-remove-key-node-same*: ‹*c* ≠ [] ⟹ *remove-key-children k c* ≠ [] ⟹
node (*hd* (*remove-key-children k c*)) = *node* (*hd c*) ⟷ *node* (*hd c*) ≠ *k*›
using *remove-key-remove-all*[*of k*]
apply (*induction k c rule*: *remove-key-children.induct*)
apply (*auto*)[]
by *fastforce*

lemma *hd-remove-key-node-same'*: ‹*c* ≠ [] ⟹ *remove-key-children k c* ≠ [] ⟹
node (*hd c*) = *node* (*hd* (*remove-key-children k c*)) ⟷ *node* (*hd c*) ≠ *k*›
using *hd-remove-key-node-same*[*of c k*] **by** *auto*

lemma *remove-key-children-node-hd*[*simp*]: ‹*c* ≠ [] ⟹ *remove-key-children* (*node* (*hd c*)) *c*= *remove-key-children*

(*node* (*hd c*)) (*tl c*)›
by (*cases c*; *cases* ‹*tl c*›; *cases* ‹*hd c*›)
(*auto simp*:)

lemma *remove-key-children-alt-def*:
‹*remove-key-children k xs* = *map* (λ*x*. *case x of Hp a b c* ⇒ *Hp a b* (*remove-key-children k c*)) (*filter* (λ*n*. *node n* ≠ *k*) *xs*)›
by (*induction k xs rule*: *remove-key-children.induct*) *auto*

lemma *not-orig-notin-remove-key*: ‹*b* ∉# *sum-list* (*map mset-nodes xs*) ⟹
b ∉# *sum-list* (*map mset-nodes* (*remove-key-children a xs*))›
by (*induction a xs rule*: *remove-key-children.induct*) *auto*

lemma *hp-parent-None-notin-same-hd*[*simp*]: ‹*b* ∉# *sum-list* (*map mset-nodes x3*) ⟹ *hp-parent b* (*Hp b x2 x3*) = *None*›
by (*induction x3 rule*: *induct-list012*)
(*auto simp*: *hp-parent-children-cons hp-parent.simps*(*1*) *filter-empty-conv split*: *if-splits*)

lemma *hp-parent-children-remove-key-children*:
‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset xs*)) ⟹ *a* ≠ *b* ⟹ *hp-parent-children b* (*remove-key-children a xs*) = *hp-parent-children b xs*›
oops

lemma *hp-parent-remove-key*:
‹*distinct-mset* ((*mset-nodes xs*)) ⟹ *a* ≠ *node xs* ⟹ *hp-parent a* (*the* (*remove-key a xs*)) = *None*›
apply (*induction a xs rule*: *remove-key.induct*)
subgoal for *a b sc children*
apply (*cases* ‹*remove-key-children a children*›)
apply (*auto simp*: *hp-parent-simps-if*)
apply (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims distinct-mset-add empty-iff hp.sel*(*1*) *inter-iff list.map*(*2*) *list.sel*(*1*) *list.simps*(*3*) *node-hd-in-sum node-in-mset-nodes set-mset-empty sum-list-simps*(*2*))
by (*metis hp-parent-children-remove-key-children mset-map sum-mset-sum-list*)
done

lemma *find-key-children-None-or-itself*[*simp*]:
‹*find-key-children a h* ≠ *None* ⟹ *node* (*the* (*find-key-children a h*)) = *a*›
by (*induction a h rule*: *find-key-children.induct*)
(*auto split*: *option.splits*)

lemma *find-key-None-or-itself*[*simp*]:
‹*find-key a h* ≠ *None* ⟹ *node* (*the* (*find-key a h*)) = *a*›
apply (*induction a h rule*: *find-key.induct*)
apply *auto*
using *find-key-children-None-or-itself* **by** *fastforce*

lemma *find-key-children-notin*[*simp*]:
‹*a* ∉# $\sum_{\#}$ (*mset-nodes* '# *mset xs*) ⟹ *find-key-children a xs* = *None*›
by (*induction a xs rule*: *find-key-children.induct*) *auto*

lemma *find-key-notin*[*simp*]:
‹*a* ∉# *mset-nodes h* ⟹ *find-key a h* = *None*›
by (*induction a h rule*: *find-key.induct*) *auto*

lemma *mset-nodes-find-key-children-subset*:
⟨*find-key-children a h* ≠ *None* ⟹ *mset-nodes* (*the* (*find-key-children a h*)) ⊆# $\sum_{\#}$ (*mset-nodes* '# *mset h*)⟩
apply (*induction a h rule*: *find-key-children.induct*)
apply (*auto split*: *option.splits simp*: *ac-simps intro*: *mset-le-incr-right*)
apply (*metis mset-le-incr-right union-commute union-mset-add-mset-right*)+
done

lemma *hp-parent-None-iff-children-None*:
⟨*hp-parent z* (*Hp x n c*) = *None* ⟷ (*c* ≠ [] ⟶ *z* ≠ *node* (*hd c*)) ∧ *hp-parent-children* (*z*) *c* = *None*⟩
by (*cases c*; *cases* ⟨*tl c*⟩)
(*auto simp*: *hp-parent-children-cons hp-parent-simps-if hp-parent.simps*(*1*) *filter-empty-conv split*: *option.splits*)

lemma *mset-nodes-find-key-subset*:
⟨*find-key a h* ≠ *None* ⟹ *mset-nodes* (*the* (*find-key a h*)) ⊆# *mset-nodes h*⟩
apply (*induction a h rule*: *find-key.induct*)
apply (*auto intro*: *mset-nodes-find-key-children-subset*)
by (*metis mset-nodes-find-key-children-subset option.distinct*(*2*) *option.sel subset-mset-imp-subset-add-mset sum-image-mset-sum-map*)

lemma *find-key-none-iff*[*simp*]:
⟨*find-key-children a h* = *None* ⟷ *a* ∉# $\sum_{\#}$ (*mset-nodes* '# *mset h*)⟩
by (*induction a h rule*: *find-key-children.induct*) *auto*

lemma *find-key-noneD*:
⟨*find-key-children a h* = *Some x* ⟹ *a* ∈# $\sum_{\#}$ (*mset-nodes* '# *mset h*)⟩
using *find-key-none-iff* **by** (*metis option.simps*(*2*))

lemma *hp-parent-children-hd-None*[*simp*]:
⟨*xs* ≠ [] ⟹ *distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset xs*)) ⟹ *hp-parent-children* (*node* (*hd xs*)) *xs* = *None*⟩
by (*cases xs*; *cases* ⟨*hd xs*⟩)
(*auto simp*: *hp-parent-children-def filter-empty-conv sum-list-map-remove1*)

lemma *hp-parent-hd-None*[*simp*]:
⟨*x* ∉# ($\sum_{\#}$ (*mset-nodes* '# *mset xs*)) ⟹*x* ∉# *sum-list* (*map mset-nodes c*) ⟹ *hp-parent-children x* (*Hp x n c* # *xs*) = *None*⟩
by (*cases xs*; *cases* ⟨*hd xs*⟩; *cases c*)
(*auto simp*: *hp-parent-children-def filter-empty-conv sum-list-map-remove1 hp-parent.simps*(*1*))

lemma *hp-parent-none-children*: ⟨*hp-parent-children z c* = *None* ⟹
hp-parent z (*Hp x n c*) = *Some x2a* ⟷ (*c* ≠ [] ∧ *z* = *node* (*hd c*) ∧ *x2a* = *Hp x n c*)⟩
by (*cases c*)
(*auto simp*: *filter-empty-conv sum-list-map-remove1 hp-parent-simps-if*)

lemma *hp-parent-children-remove-key-children*:
⟨*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset xs*)) ⟹ *a* ≠ *b* ⟹ *hp-parent-children b* (*remove-key-children a xs*) =
(*if find-key-children b xs* ≠ *None then None else hp-parent-children b xs*)⟩
oops

lemma *in-the-default-empty-iff*: ⟨*b* ∈# *the-default* {#} *M* ⟷ *M* ≠ *None* ∧ *b* ∈# *the M*⟩

by (*cases M*) *auto*

lemma *remove-key-children-hd-tl*: ‹*distinct-mset* (*sum-list* (*map mset-nodes c*)) ⟹ *c* ≠ [] ⟹ *remove-key-children* (*node* (*hd c*)) (*tl c*) = *tl c*›
by (*cases c*) (*auto simp add*: *disjunct-not-in distinct-mset-add*)

lemma *in-find-key-children-notin-remove-key*:
‹*find-key-children k c* = *Some x2* ⟹ *distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset c*)) ⟹
b ∈# *mset-nodes x2* ⟹
b ∉# $\sum_{\#}$(*mset-nodes* '# *mset* (*remove-key-children k c*))›
apply (*induction k c rule*: *remove-key-children.induct*)
subgoal by *auto*
subgoal for *k x n c xs*
using *find-key-children-None-or-itself*[*of b c*] *find-key-children-None-or-itself*[*of b xs*]
using *distinct-mset-union*[*of* ‹$\sum_{\#}$ (*mset-nodes* '# *mset xs*)› ‹$\sum_{\#}$ (*mset-nodes* '# *mset c*)›, *unfolded add.commute*[*of* ‹$\sum_{\#}$ (*mset-nodes* '# *mset xs*)›]]
distinct-mset-union[*of* ‹$\sum_{\#}$ (*mset-nodes* '# *mset c*)› ‹$\sum_{\#}$ (*mset-nodes* '# *mset xs*)›]
apply (*auto dest*: *multi-member-split*[*of b*] *split*: *option.splits*)
apply (*auto dest*!: *multi-member-split*[*of b*])[]
apply (*metis mset-nodes-find-key-children-subset option.distinct*(*1*) *option.sel subset-mset.le-iff-add sum-image-mset-sum-map union-iff*)
apply (*metis add-diff-cancel-right′ distinct-mset-in-diff mset-nodes-find-key-children-subset option.distinct*(*1*) *option.sel subset-mset.le-iff-add*
sum-image-mset-sum-map)
apply (*metis mset-nodes-find-key-children-subset mset-subset-eqD option.distinct*(*2*) *option.sel sum-image-mset-sum-m*
by (*metis add-diff-cancel-right′ distinct-mset-in-diff mset-nodes-find-key-children-subset not-orig-notin-remove-key option.distinct*(*1*)
option.sel subset-mset.le-iff-add sum-image-mset-sum-map)
done

lemma *hp-parent-children-None-hp-parent-iff*: ‹*hp-parent-children b list* = *None* ⟹ *hp-parent b* (*Hp x n list*) = *Some x2a* ⟷ *list* ≠ [] ∧ *node* (*hd list*) = *b* ∧ *x2a* = *Hp x n list*›
by (*cases list*; *cases* ‹*tl list*›) (*auto simp*: *hp-parent-simps-if*)

lemma *hp-parent-children-not-hd-node*:
‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset c*)) ⟹ *node* (*hd c*) = *node x2a* ⟹ *c* ≠ [] ⟹ *remove-key-children* (*node x2a*) *c* ≠ [] ⟹
hp-parent-children (*node* (*hd* (*remove-key-children* (*node x2a*) *c*))) *c* = *Some x2a* ⟹ *False*›
apply (*cases c*; *cases* ‹*tl c*›; *cases* ‹*hd c*›)
apply (*auto simp*: *hp-parent-children-cons*
split: *option.splits*)
apply (*simp add*: *disjunct-not-in distinct-mset-add*)
apply (*metis hp-parent-in-nodes option.distinct*(*1*) *option.sel*)
by (*smt* (*verit*, *ccfv-threshold*) *WB-List-More.distinct-mset-union2 add-diff-cancel-right′ distinct-mset-in-diff hp.sel*(*1*) *hp-parent-children-in-nodes hp-parent-simps-single-if list.sel*(*1*) *node-hd-in-sum node-in-mset-nodes option.sel option-last-Nil option-last-Some-iff*(*2*) *remove-key-children.elims sum-image-mset-sum-map*)

lemma *hp-parent-children-hd-tl-None*[*simp*]: ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset c*)) ⟹ *c* ≠ [] ⟹ *a* ∈ *set* (*tl c*)⟹ *hp-parent-children* (*node a*) *c* = *None*›
apply (*cases c*)
apply (*auto simp*: *hp-parent-children-def filter-empty-conv dest*!: *split-list*[*of a*])
apply (*metis add-diff-cancel-left′ add-diff-cancel-right′ distinct-mset-add distinct-mset-in-diff hp-parent-None-notin node-in-mset-nodes*)
apply (*simp add*: *distinct-mset-add*)
apply (*simp add*: *distinct-mset-add*)
apply (*metis* (*no-types*, *opaque-lifting*) *disjunct-not-in hp-parent-None-notin mset-subset-eqD mset-subset-eq-add-right*

node-in-mset-nodes sum-list-map-remove1 union-commute)
by (*metis WB-List-More.distinct-mset-union2 add-diff-cancel-left′ distinct-mem-diff-mset hp-parent-None-notin node-in-mset-nodes sum-list-map-remove1 union-iff*)

lemma *hp-parent-hp-parent-remove-key-not-None-same*:
assumes ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *mset c*))› **and**
‹*x* ∉# $\sum_{\#}$ (*mset-nodes* ‘# *mset c*)› **and**
‹*hp-parent b* (*Hp x n c*) = *Some x2a*› ‹*b* ∉# *mset-nodes x2a*›
‹*hp-parent b* (*Hp x n* (*remove-key-children k c*)) = *Some x2b*›
shows ‹*remove-key k x2a* ≠ *None* ∧ (*case remove-key k x2a of Some a* ⇒ (*x2b*) = *a* | *None* ⇒ *node x2a* = *k*)›
proof −
show *?thesis*
using *assms*
proof (*induction k c rule*: *remove-key-children.induct*)
case (*1 k*)
then show *?case* **by** (*auto simp*: *hp-parent-children-cons split*: *if-splits*)
next
case (*2 k x n c xs*)
moreover have ‹*c* ≠ [] ⟹ *xs* ≠ [] ⟹ *node* (*hd xs*) ≠ *node* (*hd c*)›
using *2*(*4*) **by** (*cases c*; *cases* ‹*hd c*›; *cases xs*; *auto*)
moreover have ‹*xs* ≠ [] ⟹ *hp-parent-children* (*node* (*hd xs*)) *c* = *None*›
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ calculation*(*4*) *distinct-mset-in-diff distinct-mset-union hp-parent-children-None-notin list.map*(*2*) *mset-nodes.simps node-hd-in-sum sum-image-mset-sum-map sum-list.Cons*)
moreover have ‹*c* ≠ [] ⟹ *hp-parent-children* (*node* (*hd c*)) *xs* = *None*›
by (*metis calculation*(*4*) *disjunct-not-in distinct-mset-add hp-parent-None-iff-children-None hp-parent-None-notin hp-parent-children-None-notin list.simps*(*9*) *sum-image-mset-sum-map sum-list.Cons*)
moreover have [*simp*]: ‹*remove-key* (*node x2a*) *x2a* = *None*›
by (*cases x2a*) *auto*
moreover have
‹*hp-parent-children b xs* ≠ *None* ⟹ *hp-parent-children b c* = *None*›
‹*hp-parent-children b c* ≠ *None* ⟹ *hp-parent-children b xs* = *None*›
‹*node x2a* ∈# *sum-list* (*map mset-nodes c*) ⟹ *node x2a* ∉# *sum-list* (*map mset-nodes xs*)›
using *hp-parent-children-in-nodes*[*of b xs*] *hp-parent-children-in-nodes*[*of b c*] *2*(*4*)
apply *auto*
apply (*metis disjunct-not-in distinct-mset-add hp-parent-children-None-notin option.distinct*(*1*))
apply (*metis disjunct-not-in distinct-mset-add hp-parent-children-None-notin option.distinct*(*1*))
apply (*metis disjunct-not-in distinct-mset-add hp-parent-children-None-notin option.distinct*(*1*))
done
ultimately show *?case*
using *distinct-mset-union*[*of* ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset xs*)› ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset c*)›, *unfolded add.commute*[*of* ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset xs*)›]]
distinct-mset-union[*of* ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset c*)› ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset xs*)›]
hp-parent-children-in-nodes[*of b c*] *hp-parent-children-in-nodes*[*of b xs*]
apply (*auto simp*: *hp-parent-children-cons remove-key-None-iff split*: *if-splits option.splits*)
apply (*auto simp*: *hp-parent-simps-single-if hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*auto simp*: *hp-parent-simps-single-if hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*auto simp*: *hp-parent-children-cons hp-parent-simps-single-if handy-if-lemma split*: *if-splits option.splits*)[]
apply (*cases* ‹*xs* = []›; *cases* ‹*b* = *node* (*hd xs*)›; *cases* ‹*remove-key-children* (*node x2a*) *xs* = []›;
cases ‹*b* = *node* (*hd* (*remove-key-children* (*node x2a*) *xs*))›; *cases* ‹*Hp x n* (*remove-key-children* (*node x2a*) *xs*) = *x2b*›;
auto simp: *hp-parent-children-cons hp-parent-simps-single-if handy-if-lemma split*: *if-splits op-*

tion.splits)
apply (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims disjunct-not-in distinct-mset-add hp.sel*(*1*) *hp-parent-children-hd-None list.sel*(*1*)
list.simps(*9*) *node-in-mset-nodes option-last-Nil option-last-Some-iff* (*2*) *remove-key-children-notin-unchanged sum-image-mset-sum-map sum-list.Cons*)
apply (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims disjunct-not-in distinct-mset-add hp.sel*(*1*) *hp-parent-children-hd-None list.sel*(*1*)
list.simps(*9*) *node-in-mset-nodes option-last-Nil option-last-Some-iff* (*2*) *remove-key-children-notin-unchanged sum-image-mset-sum-map sum-list.Cons*)
apply (*cases* ‹*xs* = []›; *cases* ‹*b* = *node* (*hd xs*)›; *cases* ‹*remove-key-children* (*node x2a*) *xs* = []›;
cases ‹*b* = *node* (*hd* (*remove-key-children* (*node x2a*) *xs*))›; *cases* ‹*Hp x n* (*remove-key-children* (*node x2a*) *xs*) = *x2b*›;
auto simp: *hp-parent-children-cons hp-parent-simps-single-if handy-if-lemma split*: *if-splits option.splits*)
apply (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims disjunct-not-in distinct-mset-add hp.sel*(*1*) *hp-parent-children-hd-None list.sel*(*1*)
list.simps(*9*) *node-in-mset-nodes option-last-Nil option-last-Some-iff* (*2*) *remove-key-children-notin-unchanged sum-image-mset-sum-map sum-list.Cons*)
apply (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims disjunct-not-in distinct-mset-add hp.sel*(*1*) *hp-parent-children-hd-None list.sel*(*1*)
list.simps(*9*) *node-in-mset-nodes option-last-Nil option-last-Some-iff* (*2*) *remove-key-children-notin-unchanged sum-image-mset-sum-map sum-list.Cons*)
apply (*cases* ‹*xs* = []›; *cases* ‹*b* = *node* (*hd xs*)›; *cases* ‹*remove-key-children* (*node x2a*) *xs* = []›;
cases ‹*b* = *node* (*hd* (*remove-key-children* (*node x2a*) *xs*))›; *cases* ‹*Hp x n* (*remove-key-children* (*node x2a*) *xs*) = *x2b*›;
cases ‹*node x2a* ∈# *sum-list* (*map mset-nodes c*)›; *auto simp*: *hp-parent-children-cons hp-parent-simps-single-if handy-if-lemma split*: *if-splits option.splits*)

apply (*cases* ‹*xs* = []›; *cases* ‹*b* = *node* (*hd xs*)›; *cases* ‹*remove-key-children* (*node x2a*) *xs* = []›;
cases ‹*b* = *node* (*hd* (*remove-key-children* (*node x2a*) *xs*))›; *cases* ‹*Hp x n* (*remove-key-children* (*node x2a*) *xs*) = *x2b*›;
auto simp: *hp-parent-children-cons hp-parent-simps-single-if handy-if-lemma hd-remove-key-node-same*[*of c* ‹*node x2a*›]
intro: *hp-parent-children-not-hd-node*
split: *if-splits option.splits*)

apply (*cases* ‹*xs* = []›; *cases* ‹*b* = *node* (*hd xs*)›; *cases* ‹*remove-key-children* (*node x2a*) *xs* = []›;
cases ‹*b* = *node* (*hd* (*remove-key-children* (*node x2a*) *xs*))›; *cases* ‹*Hp x n* (*remove-key-children* (*node x2a*) *xs*) = *x2b*›;
auto simp: *hp-parent-children-cons hp-parent-simps-single-if handy-if-lemma hd-remove-key-node-same*
dest: *hp-parent-children-not-hd-node split*: *if-splits option.splits*
intro: *hp-parent-children-not-hd-node*)

apply (*cases* ‹*xs* = []›; *cases* ‹*b* = *node* (*hd xs*)›; *cases* ‹*remove-key-children* (*node x2a*) *xs* = []›;
cases ‹*b* = *node* (*hd* (*remove-key-children* (*node x2a*) *xs*))›; *cases* ‹*Hp x n* (*remove-key-children* (*node x2a*) *xs*) = *x2b*›;
cases ‹*node x2a* ∈# *sum-list* (*map mset-nodes c*)›;
auto simp: *hp-parent-children-cons hp-parent-simps-single-if handy-if-lemma split*: *if-splits option.splits*)

apply (*cases* ‹*xs* = []›)
apply (*auto simp*: *hp-parent-children-cons hp-parent-simps-single-if handy-if-lemma hd-remove-key-node-same remove-key-children-hd-tl*
dest: *hp-parent-children-not-hd-node split*: *if-splits option.splits*)[*2*]
apply (*smt* (*verit*, *best*) *hd-remove-key-node-same′ hp-parent-None-iff-children-None hp-parent-children-hd-None hp-parent-children-hd-tl-None hp-parent-simps-single-if in-hd-or-tl-conv mset-map option-hd-Nil option-hd-Some-iff* (*1*)

remove-key-children-hd-tl remove-key-children-node-hd sum-mset-sum-list)
apply (*smt* (*verit*, *best*) *hd-remove-key-node-same′ hp-parent-None-iff-children-None hp-parent-children-hd-None hp-parent-children-hd-tl-None hp-parent-simps-single-if in-hd-or-tl-conv mset-map option-hd-Nil option-hd-Some-iff* (*1*) *remove-key-children-hd-tl remove-key-children-node-hd sum-mset-sum-list*)
apply (*metis add-diff-cancel-right′ distinct-mset-in-diff hp-parent-children-None-notin not-orig-notin-remove-key option-hd-Nil option-hd-Some-iff* (*2*))
apply (*metis disjunct-not-in distinct-mset-add hp-parent-children-None-notin node-hd-in-sum not-orig-notin-remove-key option-last-Nil option-last-Some-iff* (*1*) *remove-key-children-hd-tl remove-key-children-node-hd*)
apply (*smt* (*verit*, *del-insts*) *remove-key-children.simps*(*1*) *hd-remove-key-node-same′ hp-parent-children-None-notin hp-parent-children-hd-None hp-parent-children-hd-tl-None in-hd-or-tl-conv option.distinct*(*1*) *remove-key-children-hd-tl remove-key-children-node-hd remove-key-remove-all sum-image-mset-sum-map*)
apply (*metis add-diff-cancel-right′ distinct-mset-in-diff hp-parent-children-None-notin not-orig-notin-remove-key option-hd-Nil option-hd-Some-iff* (*2*))
by (*metis disjunct-not-in distinct-mset-add hp-parent-children-None-notin not-orig-notin-remove-key option-hd-Nil option-hd-Some-iff* (*2*))
qed
qed

lemma *in-remove-key-children-changed*: ‹$k \in\#$ *sum-list* (*map mset-nodes c*) $\Longrightarrow$ *remove-key-children k c* $\neq$ *c*›
apply (*induction k c rule*: *remove-key-children.induct*)
apply *auto*
apply (*metis hp.sel*(*1*) *list.sel*(*1*) *mset-map neq-Nil-conv node-hd-in-sum remove-key-remove-all sum-mset-sum-list*)+
done

lemma *hp-parent-in-nodes2*: ‹*hp-parent* (*z*) *xs* = *Some a* $\Longrightarrow$ *node a* $\in\#$ *mset-nodes xs*›
apply (*induction z xs rule*: *hp-parent.induct*)
apply (*auto simp*: *hp-parent-children-def filter-empty-conv*)
by (*metis empty-iff hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-simps*(*2*) *hp-parent-in-nodes member-add-mset mset-nodes-simps option.discI option.sel set-mset-empty sum-image-mset-sum-map sum-mset-sum-list union-iff*)

lemma *hp-parent-children-in-nodes2*: ‹*hp-parent-children z xs* = *Some a* $\Longrightarrow$ *node a* $\in\#$ $\sum_{\#}$ (*mset-nodes* '# *mset xs*)›
apply (*induction xs*)
apply (*auto simp*: *hp-parent-children-cons filter-empty-conv split*: *option.splits*)
using *hp-parent-in-nodes* **by** *fastforce*

lemma *hp-next-in-nodes2*: ‹*hp-next* (*z*) *xs* = *Some a* $\Longrightarrow$ *node a* $\in\#$ *mset-nodes xs*›
apply (*induction z xs rule*: *hp-next.induct*)
apply (*auto simp*:)
by (*metis hp-next-children.simps*(*1*) *hp-next-children-simps*(*2*) *hp-next-children-simps*(*3*) *node-in-mset-nodes option.sel*)

lemma *hp-next-children-in-nodes2*: ‹*hp-next-children* (*z*) *xs* = *Some a* $\Longrightarrow$ *node a* $\in\#$ $\sum_{\#}$ (*mset-nodes* '# *mset xs*)›
apply (*induction z xs rule*: *hp-next-children.induct*)
apply (*auto simp*: *hp-next-in-nodes2 split*: *option.splits*)
by (*metis hp-next-children-simps*(*1*) *hp-next-children-simps*(*2*) *hp-next-children-simps*(*3*) *hp-next-in-nodes2 node-in-mset-nodes option.inject*)

lemma *in-remove-key-changed*: ‹*remove-key k a* $\neq$ *None* $\Longrightarrow$ *a* = *the* (*remove-key k a*) $\longleftrightarrow$ *k* $\notin\#$ *mset-nodes a*›
apply (*induction k a rule*: *remove-key.induct*)
apply (*auto simp*: *in-remove-key-children-changed*)
by (*metis in-remove-key-children-changed*)

lemma *node-remove-key-children-in-mset-nodes*: ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*remove-key-children k c*)) ⊆# ($\sum_{\#}$ (*mset-nodes* ‘# *mset c*))›
apply (*induction k c rule*: *remove-key-children.induct*)
apply *auto*
apply (*metis mset-le-incr-right*(*2*) *union-commute union-mset-add-mset-right*)
using *subset-mset.add-mono* **by** *blast*

lemma *remove-key-children-hp-parent-children-hd-None*: ‹*remove-key-children k c* = *a* # *list* ⟹
distinct-mset (*sum-list* (*map mset-nodes c*)) ⟹
hp-parent-children (*node a*) (*a* # *list*) = *None*›
using *node-remove-key-children-in-mset-nodes*[*of k c*]
apply (*auto simp*: *hp-parent-children-def filter-empty-conv*)
apply (*meson WB-List-More.distinct-mset-mono distinct-mset-union hp-parent-itself*)
by (*metis WB-List-More.distinct-mset-mono add-diff-cancel-left′ distinct-mem-diff-mset hp-parent-None-notin node-in-mset-nodes sum-list-map-remove1 union-iff*)

lemma *hp-next-not-same-node*: ‹*distinct-mset* (*mset-nodes b*) ⟹ *hp-next x b* = *Some y* ⟹ *x* ≠ *node y*›
apply (*induction x b rule*: *hp-next.induct*)
apply *auto*
by (*metis disjunct-not-in distinct-mset-add hp-next-children-simps*(*1*) *hp-next-children-simps*(*2*) *hp-next-children-simps*(*3*) *inter-mset-empty-distrib-right node-in-mset-nodes option.sel*)

lemma *hp-next-children-not-same-node*: ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *mset c*)) ⟹ *hp-next-children x c* = *Some y* ⟹ *x* ≠ *node y*›
apply (*induction x c rule*: *hp-next-children.induct*)
subgoal
apply (*auto simp*: *hp-next-children.simps*(*1*) *split*: *if-splits option.splits dest*: *hp-next-not-same-node*)
apply (*metis* (*no-types*, *opaque-lifting*) *distinct-mset-iff hp.exhaust-sel mset-nodes-simps union-mset-add-mset-left union-mset-add-mset-right*)
apply (*metis Duplicate-Free-Multiset.distinct-mset-mono mset-subset-eq-add-left union-commute*)
by (*meson distinct-mset-union hp-next-not-same-node*)
subgoal apply *auto*
by (*meson hp-next-not-same-node*)
subgoal by *auto*
done

lemma *hp-next-children-hd-is-hd-tl*: ‹*c* ≠ [] ⟹ *distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *mset c*)) ⟹ *hp-next-children* (*node* (*hd c*)) *c* = *option-hd* (*tl c*)›
by (*cases c*; *cases* ‹*tl c*›) *auto*

lemma *hp-parent-children-remove-key-children-other*:
assumes ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *mset xs*))›
shows ‹*hp-parent-children b* (*remove-key-children a xs*) =
(*if b* ∈# (*the-default* {#} (*map-option mset-nodes* (*find-key-children a xs*))) *then None*
else if map-option node (*hp-next-children a xs*) = *Some b then map-option* (*the o remove-key a*) (*hp-parent-children a xs*)
else map-option (*the o remove-key a*) (*hp-parent-children b xs*))›
using *assms*
proof (*induction a xs rule*: *remove-key-children.induct*)
case (*1 k*)
then show *?case* **by** *auto*
next
case (*2 k x n c xs*)

have [*intro*]: ‹*b* ∈# *sum-list* (*map mset-nodes c*) ⟹ *hp-parent-children b xs* = *None*›
using *2*(*4*) **by** (*auto simp*: *in-the-default-empty-iff dest*!: *multi-member-split split*: *if-splits*)
consider
(*kx*) ‹*k*=*x*› |
(*inc*) ‹*k* ≠ *x*› ‹*find-key-children k c* ≠ *None*› |
(*inxs*) ‹*k* ≠ *x*› ‹*find-key-children k c* = *None*›
by *blast*
then show *?case*
proof (*cases*)
case *kx*
then show *?thesis*
apply (*auto simp*: *in-the-default-empty-iff*)
using *find-key-children-None-or-itself*[*of b c*] *find-key-children-None-or-itself*[*of b xs*] *2*
using *distinct-mset-union*[*of* ‹∑# (*mset-nodes* ‘# *mset xs*)› ‹∑# (*mset-nodes* ‘# *mset c*)›, *unfolded add.commute*[*of* ‹∑# (*mset-nodes* ‘# *mset xs*)›]]
distinct-mset-union[*of* ‹∑# (*mset-nodes* ‘# *mset c*)› ‹∑# (*mset-nodes* ‘# *mset xs*)›]
by (*auto simp*: *not-orig-notin-remove-key in-the-default-empty-iff hp-parent-children-cons*
split: *option.split if-splits*)
next
case *inc*
moreover have ‹*b* ∈# *mset-nodes* (*the* (*find-key-children k c*)) ⟹ *b* ∈# ∑# (*mset-nodes* ‘# *mset c*)›
using *inc* **by** (*meson mset-nodes-find-key-children-subset mset-subset-eqD*)
moreover have *c*: ‹*c* ≠ []›
using *inc* **by** *auto*
moreover have [*simp*]: ‹*remove-key-children* (*node* (*hd c*)) (*tl c*) = *tl c*›
using *c 2*(*4*) **by** (*cases c*; *cases* ‹*hd c*›) *auto*
moreover have [*simp*]: ‹*find-key-children* (*node* (*hd c*)) *c* = *Some* (*hd c*)›
using *c 2*(*4*) **by** (*cases c*; *cases* ‹*hd c*›) *auto*
moreover have [*simp*]: ‹*k* ∈# *sum-list* (*map mset-nodes c*) ⟹ *k* ∉# *sum-list* (*map mset-nodes xs*)› **for** *k*
using *2*(*4*) **by** (*auto dest*!: *multi-member-split*)
moreover have *KK*[*iff*]: ‹*remove-key-children k c* = [] ⟷ *c* = [] ∨ (*c* ≠ [] ∧ *tl c* = [] ∧ *node* (*hd c*) = *k*)›
using *2*(*4*)
by (*induction k c rule*: *remove-key-children.induct*) (*auto simp*: *dest*: *multi-member-split*)
ultimately show *?thesis*
using *find-key-children-None-or-itself*[*of b c*] *find-key-children-None-or-itself*[*of b xs*] *2*
find-key-children-None-or-itself[*of k c*] *find-key-children-None-or-itself*[*of k xs*]
using *distinct-mset-union*[*of* ‹∑# (*mset-nodes* ‘# *mset xs*)› ‹∑# (*mset-nodes* ‘# *mset c*)›, *unfolded add.commute*[*of* ‹∑# (*mset-nodes* ‘# *mset xs*)›]]
distinct-mset-union[*of* ‹∑# (*mset-nodes* ‘# *mset c*)› ‹∑# (*mset-nodes* ‘# *mset xs*)›]
apply (*auto simp*: *not-orig-notin-remove-key in-the-default-empty-iff split*: *option.split if-splits*)
apply (*auto simp*: *hp-parent-children-cons in-the-default-empty-iff split*: *option.split*
dest: *in-find-key-children-notin-remove-key*)[]
apply (*metis hp-parent-None-iff-children-None in-find-key-children-notin-remove-key mset-map node-hd-in-sum sum-mset-sum-list*)
apply (*auto simp*: *hp-parent-children-cons in-the-default-empty-iff split*: *option.split*
dest: *in-find-key-children-notin-remove-key*)[]
apply (*metis hp-parent-None-iff-children-None in-find-key-children-notin-remove-key mset-map node-hd-in-sum sum-mset-sum-list*)
defer

apply (*auto simp*: *hp-parent-children-cons in-the-default-empty-iff hp-parent-None-iff-children-None*
hp-parent-children-None-hp-parent-iff split: *option.split*
dest: *in-find-key-children-notin-remove-key*)[]

apply (*metis KK* ‹*remove-key-children* (*node* (*hd c*)) (*tl c*) = *tl c*›
hd-remove-key-node-same′ *hp-next-children-hd-simps list.exhaust-sel option-hd-def remove-key-children-node-hd*)
apply (*metis KK hd-remove-key-node-same*′)
apply (*metis KK find-key-children-None-or-itself hd-remove-key-node-same inc*(*2*) *node-in-mset-nodes option.sel*)
apply (*smt* (*verit*) *remove-key.simps* ‹*remove-key-children* (*node* (*hd c*)) (*tl c*) = *tl c*› ‹*b* ∈# *sum-list* (*map mset-nodes c*) ⟹ *hp-parent-children b xs* = *None*›
hd-remove-key-node-same hp-next-children.elims hp-parent-None-iff-children-None hp-parent-children-hd-None hp-parent-none-children
hp-parent-simps-single-if list.sel(*1*) *list.sel*(*3*) *o-apply option.map-sel option.sel remove-key-children-node-hd sum-image-mset-sum-map*)

apply (*auto simp*: *hp-parent-children-cons in-the-default-empty-iff hp-parent-None-iff-children-None*
hp-parent-children-None-hp-parent-iff in-remove-key-changed split: *option.split*
dest: *in-find-key-children-notin-remove-key*)[]
apply (*metis* ‹*b* ∈# *sum-list* (*map mset-nodes c*) ⟹ *hp-parent-children b xs* = *None*› *hp-next-children-in-nodes2 sum-image-mset-sum-map*)

apply (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims WB-List-More.distinct-mset-mono* ‹*find-key-children* (*node* (*hd c*)) *c* = *Some* (*hd c*)› ‹*remove-key-children* (*node* (*hd c*)) (*tl c*) = *tl c*› *add-diff-cancel-left*′ *distinct-mset-in-diff hp.exhaust-sel hp.inject hp-next-children-in-nodes2*
hp-next-children-simps(*2*) *hp-next-children-simps*(*3*) *hp-next-simps in-remove-key-children-changed list.exhaust-sel list.sel*(*1*) *mset-nodes-simps*
mset-nodes-find-key-children-subset option.sel option-last-Nil option-last-Some-iff(*2*) *sum-image-mset-sum-map sum-mset-sum-list union-single-eq-member*)
apply (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims WB-List-More.distinct-mset-mono* ‹*find-key-children* (*node* (*hd c*)) *c* = *Some* (*hd c*)›
‹*remove-key-children* (*node* (*hd c*)) (*tl c*) = *tl c*› *add-diff-cancel-left*′ *distinct-mset-in-diff hp.exhaust-sel hp.inject hp-next-children-in-nodes2*
hp-next-children-simps(*2*) *hp-next-children-simps*(*3*) *hp-next-simps in-remove-key-children-changed list.exhaust-sel list.sel*(*1*) *mset-nodes-simps*
mset-nodes-find-key-children-subset option.sel option-last-Nil option-last-Some-iff(*2*) *sum-image-mset-sum-map sum-mset-sum-list union-single-eq-member*)
apply (*metis handy-if-lemma hp-next-children-hd-simps list.exhaust-sel option-hd-def*)
apply (*metis hp-next-children-hd-is-hd-tl mset-map option-hd-Some-iff*(*1*) *sum-mset-sum-list*)
by (*smt* (*verit*, *best*) *None-eq-map-option-iff remove-key.simps hp-parent-None-iff-children-None hp-parent-children-hd-None hp-parent-none-children hp-parent-simps-single-if list.exhaust-sel o-apply option.map-sel option.sel remove-key-children-hp-parent-children-hd-None sum-image-mset-sum-map*)
next
case *inxs*
moreover have *True*
by *auto*
ultimately show *?thesis*
using *find-key-children-None-or-itself*[*of b c*] *find-key-children-None-or-itself*[*of b xs*] *2*
find-key-children-None-or-itself[*of k c*] *find-key-children-None-or-itself*[*of k xs*]
hp-next-children-in-nodes2[*of k xs*]
using *distinct-mset-union*[*of* ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset xs*)› ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset c*)›, *unfolded add.commute*[*of* ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset xs*)›]]
distinct-mset-union[*of* ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset c*)› ‹$\sum_{\#}$ (*mset-nodes* ‘# *mset xs*)›]
apply (*auto simp*: *not-orig-notin-remove-key in-the-default-empty-iff split*: *option.split if-splits*)
apply (*auto simp*: *hp-parent-children-cons in-the-default-empty-iff ex-hp-node-children-Some-in-mset-nodes*
split: *option.split intro*!: *hp-parent-None-notin*
dest: *in-find-key-children-notin-remove-key multi-member-split*
mset-nodes-find-key-children-subset)[*2*]
apply (*cases* ‹*hp-next-children k xs*›)

apply (*auto simp*: *hp-parent-children-cons in-the-default-empty-iff ex-hp-node-children-Some-in-mset-nodes*
split: *option.split intro*!: *hp-parent-None-notin*
dest: *in-find-key-children-notin-remove-key multi-member-split*
mset-nodes-find-key-children-subset)[*2*]
apply (*metis* (*no-types*, *lifting*) *find-key-none-iff mset-map*
mset-nodes-find-key-children-subset mset-subset-eqD option.map-sel sum-mset-sum-list)
apply (*metis* (*no-types*, *lifting*) *find-key-none-iff mset-map*
mset-nodes-find-key-children-subset mset-subset-eqD option.map-sel sum-mset-sum-list)
apply (*metis* (*no-types*, *lifting*) *distinct-mset-add find-key-none-iff in-the-default-empty-iff inter-iff*
mset-map
mset-nodes-find-key-children-subset mset-subset-eqD option.map-sel sum-mset-sum-list the-default.simps(*2*))
apply (*smt* (*verit*) *add-diff-cancel-left′ distinct-mem-diff-mset hp-next-children-in-nodes2 hp-parent-None-iff-children-*
hp-parent-children-None-notin hp-parent-children-append-case hp-parent-children-append-skip-first hp-parent-children-cons
mset-map node-hd-in-sum sum-mset-sum-list)
apply (*auto simp*: *hp-parent-children-cons in-the-default-empty-iff split*: *option.split*
dest: *in-find-key-children-notin-remove-key*)[]
apply (*metis* (*mono-tags*, *opaque-lifting*) *remove-key.simps comp-def hp-parent-None-iff-children-None*
hp-parent-children-None-hp-parent-iff hp-parent-simps-single-if option.map-sel option.sel remove-key-children-notin-uncha
apply (*auto simp*: *hp-parent-children-cons in-the-default-empty-iff split*: *option.split*
dest: *in-find-key-children-notin-remove-key*)[]
by (*metis* (*no-types*, *opaque-lifting*) *remove-key.simps hp-parent-simps-single-if option.distinct*(*1*)
option.map(*2*) *option.map-comp option.sel remove-key-children-notin-unchanged*)
qed
qed

lemma *hp-parent-remove-key-other*:
assumes ‹*distinct-mset* ((*mset-nodes xs*))› ‹(*remove-key a xs*) ≠ *None*›
shows ‹*hp-parent b* (*the* (*remove-key a xs*)) =
(*if b* ∈# (*the-default* {#} (*map-option mset-nodes* (*find-key a xs*))) *then None*
else if map-option node (*hp-next a xs*) = *Some b then map-option* (*the o remove-key a*) (*hp-parent a*
xs)
else map-option (*the o remove-key a*) (*hp-parent b xs*))›
using *assms hp-parent-children-remove-key-children-other*[*of* ‹*hps xs*› *b a*]
apply (*cases xs*)
apply (*auto simp*: *in-the-default-empty-iff hp-parent-None-iff-children-None*
dest: *in-find-key-children-notin-remove-key split*: *if-splits*)
apply (*metis* (*no-types*, *lifting*) *in-find-key-children-notin-remove-key node-hd-in-sum sum-image-mset-sum-map*)
apply (*metis hp-parent-None-notin-same-hd hp-parent-simps-single-if in-find-key-children-notin-remove-key*
option.simps(*2*) *sum-image-mset-sum-map*)
apply (*smt* (*verit*, *ccfv-threshold*) *None-eq-map-option-iff remove-key.simps remove-key-children.elims*
distinct-mset-add distinct-mset-add-mset
hd-remove-key-node-same hp.sel(*1*) *hp-child.simps*(*1*) *hp-child-hd hp-next-children-hd-is-hd-tl hp-next-children-in-nodes*
hp-next-children-simps(*2*)
hp-next-children-simps(*3*) *hp-next-simps hp-parent-None-iff-children-None hp-parent-simps-single-if*
inter-iff list.simps(*9*) *mset-nodes-simps*
node-in-mset-nodes o-apply option.collapse option.map-sel option.sel option-hd-Some-iff(*2*) *remove-key-children-hd-tl*
remove-key-children-node-hd
sum-image-mset-sum-map sum-list-simps(*2*) *sum-mset-sum-list union-single-eq-member*)
apply (*simp add*: *hp-parent-simps-single-if*; *fail*)
apply (*simp add*: *hp-parent-simps-single-if*; *fail*)
apply (*smt* (*verit*) *find-key-children.elims remove-key.simps remove-key-children.elims find-key-none-iff*
hp.sel(*1*) *hp-next-children-hd-is-hd-tl*
hp-parent-children-None-notin hp-parent-simps-single-if list.sel(*1*) *list.sel*(*3*) *list.simps*(*3*) *map-option-eq-Some*
node-in-mset-nodes o-apply option.map-sel
option.sel option-hd-def remove-key-children-hd-tl sum-image-mset-sum-map)

apply (*simp add*: *hp-parent-simps-single-if*)
apply (*simp add*: *hp-parent-simps-single-if*)
done

lemma *hp-prev-in-nodes*: ‹*hp-prev k c≠ None ⟹ node* (*the* (*hp-prev k c*)) ∈# ((*mset-nodes c*))›
by (*induction k c rule*: *hp-prev.induct*) (*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits*)

lemma *hp-prev-children-in-nodes*: ‹*hp-prev-children k c≠ None ⟹ node* (*the* (*hp-prev-children k c*)) ∈# (∑# (*mset-nodes ‘# mset c*))›
apply (*induction k c rule*: *hp-prev-children.induct*)
subgoal for *a x y children*
using *hp-prev-in-nodes*[*of a x*]
by (*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits*)
subgoal for *a x*
using *hp-prev-in-nodes*[*of a x*]
by (*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits*)
subgoal by *auto*
done

lemma *hp-next-children-notin-end*:
‹*distinct-mset* (∑# (*mset-nodes ‘# mset* (*x#xs*))) ⟹ *hp-next-children a xs* = *None* ⟹ *hp-next-children a* (*x # xs*) = (*if a = node x then option-hd xs else hp-next a x*)›
by (*cases xs*)
(*auto simp*: *hp-next-children.simps*(*1*) *split*: *option.splits*)

lemma *hp-next-children-remove-key-children-other*:
fixes *xs* :: (*'b*, *'a*) *hp list*
assumes ‹*distinct-mset* (∑# (*mset-nodes ‘# mset xs*))›
shows ‹*hp-next-children b* (*remove-key-children a xs*) =
(*if b* ∈# (*the-default* {#} (*map-option mset-nodes* (*find-key-children a xs*))) *then None*
else if map-option node (*hp-prev-children a xs*) = *Some b then* (*hp-next-children a xs*)
else map-option (*the o remove-key a*) (*hp-next-children b xs*))›
using *assms*
proof (*induction a xs rule*: *remove-key-children.induct*)
case (*1 k*)
then show *?case* **by** *auto*
next
case (*2 k x n c xs*)
have *dist-c-rem-y-xs*: ‹*distinct-mset*
(*sum-list* (*map mset-nodes c*) +
sum-list (*map mset-nodes* (*remove-key-children y xs*)))› **for** *y*
by (*smt* (*verit*, *del-insts*) *2*(*4*) *distinct-mset-add inter-mset-empty-distrib-right mset.simps*(*2*)
mset-nodes.simps node-remove-key-children-in-mset-nodes subset-mset.add-diff-inverse
sum-image-mset-sum-map sum-mset.insert union-ac(*2*))
have ‹*distinct-mset*
(*sum-list* (*map mset-nodes c*) + *sum-list* (*map mset-nodes* (*remove-key-children k xs*)))›
‹*x* ∉# *sum-list* (*map mset-nodes* (*remove-key-children k xs*))›
using *2*(*4*) **apply** *auto*
apply (*metis distinct-mset-mono′ mset-map mset-subset-eq-mono-add-left-cancel node-remove-key-children-in-mset-no*
sum-mset-sum-list)
by (*simp add*: *not-orig-notin-remove-key*)
moreover have ‹*distinct-mset* (*sum-list*
(*map mset-nodes* (*Hp x n* (*remove-key-children k c*) # *remove-key-children k xs*)))›
using *2*(*4*) **apply** (*auto simp*: *not-orig-notin-remove-key*)
by (*metis calculation*(*1*) *distinct-mset-mono′ mset-map node-remove-key-children-in-mset-nodes*
subset-mset.add-right-mono sum-mset-sum-list)

moreover have ‹*hp-prev-children k xs* ≠ *None* ⟹ *remove-key-children k xs* ≠ []›
  using *2*(*4*) **by** (*cases xs*; *cases* ‹*hd xs*›; *cases* ‹*tl xs*›) (*auto*)
moreover have ‹*x* = *node z* ⟹ *hp-prev-children k* (*Hp* (*node z*) *n c* # *xs*) = *Some z* ⟷
  z = *Hp x n c* ∧ *xs* ≠ [] ∧ *k* = *node* (*hd* (*xs*))› **for** *z*
  using *2*(*4*) *hp-prev-children-in-nodes*[*of - c*] **apply** −
  apply (*cases* ‹*xs*›; *cases z*; *cases* ‹*hd xs*›)
  using *hp-prev-children-in-nodes*[*of - c*] **apply** *fastforce*
  apply (*auto simp*:)
apply (*metis 2*(*4*) *hp.inject hp.sel*(*1*) *hp-prev-children-in-nodes hp-prev-children-simps*(*1*) *hp-prev-children-simps*(*2*) *hp-prev-children-simps*(*3*) *hp-prev-simps list.distinct*(*1*) *list.sel*(*1*) *list.sel*(*3*) *option.sel remove-key-children-hd-tl remove-key-remove-all sum-image-mset-sum-map*)
apply (*metis 2*(*4*) *hp.inject hp.sel*(*1*) *hp-prev-children-in-nodes hp-prev-children-simps*(*1*) *hp-prev-children-simps*(*2*) *hp-prev-children-simps*(*3*) *hp-prev-simps in-remove-key-children-changed list.distinct*(*2*) *list.sel*(*1*) *list.sel*(*3*) *option.sel remove-key-children-hd-tl sum-image-mset-sum-map*)
by (*metis 2*(*4*) *hp.sel*(*1*) *hp-prev-children-in-nodes hp-prev-children-simps*(*2*) *hp-prev-children-simps*(*3*) *hp-prev-simps in-remove-key-children-changed list.distinct*(*2*) *list.sel*(*1*) *list.sel*(*3*) *option.sel remove-key-children-hd-tl sum-image-mset-sum-map*)
moreover have ‹*xs* ≠ [] ⟹ *find-key-children* (*node* (*hd xs*)) *xs* = *Some* (*hd xs*)›
  by (*metis find-key-children.simps*(*2*) *hp.exhaust-sel list.exhaust-sel*)
moreover have ‹*find-key-children k c* = *Some y* ⟹
  option-hd (*remove-key-children k xs*) =
  map-option (λ*a*. *the* (*remove-key k a*)) (*option-hd xs*)› **for** *y*
  using *2*(*4*) **by** (*cases xs*; *cases* ‹*hd xs*›) *auto*
moreover have ‹*find-key-children k c* = *Some x2* ⟹ *k* ∉# $\sum_{\#}$ (*mset-nodes* '# *mset xs*)› **for** *x2*
  by (*metis* (*no-types*, *lifting*) *2*(*4*) *Un-iff add-diff-cancel-left*′ *distinct-mset-in-diff find-key-noneD list.simps*(*9*) *mset-nodes.simps set-mset-union sum-image-mset-sum-map sum-list-simps*(*2*))
moreover have ‹*k* ∉# *sum-list* (*map mset-nodes xs*) ⟹ *k* ≠ *x* ⟹
  ∀ *za*. *hp-prev-children k* (*Hp x n c* # *xs*) = *Some za* ⟶ *node za* ≠ *node z* ⟹
  hp-prev-children k c = *Some z* ⟹ *hp-next-children* (*node z*) (*Hp x n* (*remove-key-children k c*) # *xs*) =
  map-option (λ*a*. *the* (*remove-key k a*)) (*hp-next-children* (*node z*) (*Hp x n c* # *xs*))› **for** *z*
by (*metis hp-prev-children-None-notin hp-prev-children-first-child option-last-Nil option-last-Some-iff* (*2*) *sum-image-mset-sum-map*)
moreover have ‹*find-key-children k c* = *Some x2* ⟹
  b ∈# *mset-nodes x2* ⟹
  b∉# $\sum_{\#}$ (*mset-nodes* '# *mset* (*Hp x n* (*remove-key-children k c*) # *xs*))› **for** *x2*
  by (*smt* (*verit*, *ccfv-threshold*) *2*(*4*) *add-diff-cancel-right*′ *distinct-mset-add distinct-mset-in-diff find-key-noneD find-key-none-iff in-find-key-children-notin-remove-key mset.simps*(*2*) *mset-left-cancel-union mset-nodes.simps mset-nodes-find-key-children-subset mset-subset-eqD mset-subset-eq-add-right option.sel sum-mset.insert*)
moreover have ‹ *find-key-children k c* = *Some x2* ⟹ *k* ≠ *x* ⟹ *k* ∈# *sum-list* (*map mset-nodes c*)› **for** *x2*
  by (*metis find-key-none-iff option.distinct*(*1*) *sum-image-mset-sum-map*)
moreover have [*simp*]: ‹*z* ∈# *sum-list* (*map mset-nodes c*) ⟹ *hp-next-children* (*z*) *xs* = *None*› **for** *z*
  using *2.prems distinct-mset-in-diff* **by** *fastforce*
moreover have ‹∀ *z*. *hp-prev-children k* (*Hp x n c* # *xs*) = *Some z* ⟶ *node z* ≠ *x* ⟹
  xs = [] ∨ (*xs* ≠ [] ∧ *node* (*hd xs*) ≠ *k*)›
  by (*smt* (*verit*, *ccfv-SIG*) *remove-key.simps remove-key-children.elims hp.sel*(*1*) *hp-prev-children-simps*(*1*) *list.sel*(*1*) *list.simps*(*3*) *option.map*(*1*) *option.map*(*2*) *option.sel option-hd-Nil option-hd-Some-hd*)
moreover have ‹*xs* ≠ [] ⟹ *node* (*hd xs*) ≠ *k* ⟹ *remove-key-children k xs* ≠ []› **and**
  [*simp*]: ‹*xs* ≠ [] ⟹ *node* (*hd xs*) ≠ *k* ⟹ *hd* (*remove-key-children k xs*) = *the* (*remove-key k* (*hd xs*))›
  ‹*xs* ≠ [] ⟹ *node* (*hd xs*) ≠ *k* ⟹ *k* ∉# *sum-list* (*map mset-nodes c*) ⟹ *hp-prev-children k* (*Hp x n c* # *xs*) = *Some z* ⟷ *hp-prev-children k* (*xs*) =*Some z*›

‹k ∉# sum-list (map mset-nodes xs) ⟹ xs ≠ [] ⟹ the (remove-key k (hd xs)) = hd xs›
for z
by (cases xs; cases ‹hd xs›; auto; fail)+
moreover have‹mset-nodes y ⊆# sum-list (map mset-nodes xs)⟹
distinct-mset (sum-list (map mset-nodes c) + sum-list (map mset-nodes xs)) ⟹ b ∈# mset-nodes y ⟹
b ∉# (sum-list (map mset-nodes c))› for y :: ('b, 'a) hp
by (metis (no-types, lifting) add-diff-cancel-right' distinct-mset-in-diff mset-subset-eqD)
moreover have ‹hp-prev-children k xs = Some z ⟹ hp-next-children (node z) c = None› for z
using distinct-mset-union[of ‹∑# (mset-nodes '# mset xs)› ‹∑# (mset-nodes '# mset c)›, unfolded add.commute[of ‹∑# (mset-nodes '# mset xs)›]]
distinct-mset-union[of ‹∑# (mset-nodes '# mset c)› ‹∑# (mset-nodes '# mset xs)›]
by (metis 2.prems disjunct-not-in dist-c-rem-y-xs distinct-mset-add hp-next-children-None-notin
hp-prev-children-in-nodes list.sel(3) list.simps(3) option.sel option.simps(2) remove-key-children-hd-tl sum-image-mset-sum-map)
ultimately show ?case
using distinct-mset-union[of ‹∑# (mset-nodes '# mset xs)› ‹∑# (mset-nodes '# mset c)›, unfolded add.commute[of ‹∑# (mset-nodes '# mset xs)›]]
distinct-mset-union[of ‹∑# (mset-nodes '# mset c)› ‹∑# (mset-nodes '# mset xs)›] 2
find-key-children-None-or-itself[of k c] find-key-children-None-or-itself[of k xs] hp-prev-children-in-nodes[of b c]
hp-prev-children-in-nodes[of k c] mset-nodes-find-key-children-subset[of k xs]
supply [simp del] = find-key-children-None-or-itself
apply (auto split: option.splits if-splits simp: remove-key-children-hd-tl comp-def
in-the-default-empty-iff)
apply (simp add: disjunct-not-in distinct-mset-add)
apply (auto simp: hp-next-children-simps-if remove-key-children-hd-tl
dist-c-rem-y-xs hp-next-children-notin-end
hp-next-children-hd-is-hd-tl
split: option.splits)
by (metis (no-types, lifting) 2.prems remove-key-children.simps(1)
hp-prev-children-None-notin hp-prev-children-skip-Cons hp-prev-in-nodes hp-prev-skip-hd-children
list.exhaust-sel mset.simps(2) node-in-mset-nodes option.map-sel option.sel option-last-Nil
option-last-Some-iff(2) remove-key-children-hd-tl remove-key-remove-all sum-image-mset-sum-map sum-mset.insert union-commute)
qed

lemma *hp-next-remove-key-other*:
assumes ‹distinct-mset (mset-nodes xs)› ‹remove-key a xs ≠ None›
shows ‹hp-next b (the (remove-key a xs)) =
(if b ∈# (the-default {#} (map-option mset-nodes (find-key a xs))) then None
else if map-option node (hp-prev a xs) = Some b then (hp-next a xs)
else map-option (the o remove-key a) (hp-next b xs))›
using hp-next-children-remove-key-children-other[of ‹hps xs› b a] assms
by (cases xs) (auto)

lemma *hp-prev-children-cons-if*:
‹hp-prev-children b (a # xs) = (if map-option node (option-hd xs) = Some b then Some a
else (case hp-prev-children b (hps a) of None ⇒ hp-prev-children b xs | Some a ⇒ Some a))›
apply (cases xs)
apply (auto split: option.splits simp: hp-prev-children.simps(1))
apply (metis hp.collapse hp-prev-simps)
apply (metis hp.exhaust-sel hp-prev-simps)
apply (metis hp.exhaust-sel hp-prev-simps option.simps(2))
apply (metis hp.exhaust-sel hp-prev-simps option.simps(2))

by (*metis hp.exhaust-sel hp-prev-simps the-default.simps*(*1*))

lemma *hp-prev-children-remove-key-children-other*:
assumes ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset xs*))›
shows ‹*hp-prev-children b* (*remove-key-children a xs*) =
(*if b* ∈# (*the-default* {#} (*map-option mset-nodes* (*find-key-children a xs*))) *then None*
else if map-option node (*hp-next-children a xs*) = *Some b then* (*hp-prev-children a xs*)
else map-option (*the o remove-key a*) (*hp-prev-children b xs*))›
using *assms*
proof (*induction a xs rule*: *remove-key-children.induct*)
case (*1 k*)
then show *?case* **by** *auto*
next
case (*2 k x n c xs*)
have *find-None-not-other*: ‹*find-key-children k c* ≠ *None* ⟹ *find-key-children k xs* = *None*›
‹*find-key-children k xs* ≠ *None* ⟹ *find-key-children k c* = *None*›
using *2*(*4*) *distinct-mset-in-diff* **apply** *fastforce*
using *2*(*4*) *distinct-mset-in-diff* **by** *fastforce*

have [*simp*]: ‹*remove-key-children k xs* ≠ [] ⟹ *xs* ≠ []›
by *auto*

have [*simp*]: ‹*hp-prev-children* (*node* (*hd xs*)) *xs* = *None*›
using *2*(*4*)
by (*cases xs*; *cases hd xs*; *cases tl xs*)
auto
have *remove-key-children-empty-iff*: ‹(*remove-key-children k xs* = []) = (∀ *x*. *x* ∈ *set xs* ⟶ *node x* = *k*)›
by (*auto simp*: *remove-key-children-alt-def filter-empty-conv*)
have [*simp*]: ‹*find-key-children k c* = *Some x2* ⟹ *remove-key-children k xs* = *xs*› **for** *x2*
by (*metis* ‹*find-key-children k c* ≠ *None* ⟹ *find-key-children k xs* = *None*› *find-key-none-iff option.distinct*(*1*) *remove-key-children-notin-unchanged sum-image-mset-sum-map*)

have *dist*: ‹*distinct-mset*
(*sum-list* (*map mset-nodes c*) + *sum-list* (*map mset-nodes* (*remove-key-children k xs*)))›
‹*x* ∉# *sum-list* (*map mset-nodes* (*remove-key-children k xs*))›
using *2*(*4*) **apply** *auto*
apply (*metis distinct-mset-mono′ mset-map mset-subset-eq-mono-add-left-cancel node-remove-key-children-in-mset-no
sum-mset-sum-list)
by (*simp add*: *not-orig-notin-remove-key*)
moreover have ‹*distinct-mset*
(*sum-list*
(*map mset-nodes* (*Hp x n* (*remove-key-children k c*) # *remove-key-children k xs*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* (*remove-key-children k xs*)))›
using *2*(*4*) **apply** (*auto simp*: *not-orig-notin-remove-key*)
apply (*metis dist*(*1*) *distinct-mset-mono′ mset-map node-remove-key-children-in-mset-nodes subset-mset.add-right-mono sum-mset-sum-list*)
using *WB-List-More.distinct-mset-union2 calculation* **by** *blast*
moreover have ‹*hp-next-children k xs* ≠ *None* ⟹ *remove-key-children k xs* ≠ []›
using *2*(*4*) **by** (*cases xs*; *cases* ‹*hd xs*›; *cases* ‹*tl xs*›) (*auto*)
moreover have ‹*xs* ≠ [] ⟹ *find-key-children* (*node* (*hd xs*)) *xs* = *Some* (*hd xs*)›
by (*metis find-key-children.simps*(*2*) *hp.exhaust-sel list.exhaust-sel*)
moreover have [*simp*]: ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset a*)) ⟹ *hp-prev-children* (*node* (*hd a*)) *a* = *None*› **for** *a*
by (*cases* ‹(*node* (*hd a*), *a*)› *rule*: *hp-prev-children.cases*; *cases* ‹*hd a*›)

(*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits*)
moreover have
‹*remove-key-children k xs* ≠ [] ⟹ *hp-prev-children* (*node* (*hd* (*remove-key-children k xs*))) (*remove-key-children k c*) = *None*›
‹*xs* ≠ [] ⟹ *hp-prev-children* (*node* (*hd xs*)) (*remove-key-children k c*) = *None*›
apply (*metis dist*(*1*) *disjunct-not-in distinct-mset-add hp-prev-children-None-notin node-hd-in-sum not-orig-notin-remove-key sum-image-mset-sum-map*)
by (*smt* (*verit*, *ccfv-threshold*) *remove-key-children.elims add-diff-cancel-right*′ *dist*(*1*) *distinct-mem-diff-mset hp.sel*(*1*)
hp-prev-children-None-notin list.distinct(*2*) *list.sel*(*1*) *mset-subset-eqD node-hd-in-sum node-remove-key-children-in-remove-key-remove-all sum-image-mset-sum-map*)
have ‹*hp-next-children k c* = *Some z* ⟹
hp-prev-children (*node z*) (*Hp x n* (*remove-key-children k c*) # *remove-key-children k xs*) =
hp-prev-children (*node z*) (*remove-key-children k c*)› **for** *z*
apply (*auto simp*: *hp-prev-children-cons-if split*: *option.splits simp del*:)
apply (*metis add-diff-cancel-right*′ *calculation*(*1*) *distinct-mset-in-diff hp-next-children-in-nodes2 node-hd-in-sum sum-image-mset-sum-map*)
apply (*metis add-diff-cancel-right*′ *calculation*(*1*) *distinct-mset-in-diff hp-next-children-in-nodes2 hp-prev-children-None-notin sum-image-mset-sum-map*)
by (*metis* ‹*remove-key-children k xs* ≠ [] ⟹ *hp-prev-children* (*node* (*hd* (*remove-key-children k xs*))) (*remove-key-children k c*) = *None*› *option.simps*(*2*))
moreover have ‹*b* ∈# *sum-list* (*map mset-nodes c*) ⟹ *hp-prev-children b xs* = *None*› **for** *b*
by (*metis* (*no-types*, *lifting*) *2*(*4*) *add-diff-cancel-left*′ *distinct-mset-add distinct-mset-in-diff hp-prev-children-None-no*
list.map(*2*) *mset-nodes-simps sum-image-mset-sum-map sum-list.Cons*)

moreover have ‹*find-key-children k c* ≠ *None* ⟹ *xs* ≠ [] ⟹ *node* (*hd xs*) ∉# *mset-nodes* (*the* (*find-key-children k c*))›
by (*metis* (*no-types*, *opaque-lifting*) ‹⋀*x2*. *find-key-children k c* = *Some x2* ⟹ *remove-key-children k xs* = *xs*›
add-diff-cancel-right′ *calculation*(*1*) *calculation*(*6*) *distinct-mset-in-diff mset-nodes-find-key-children-subset*
mset-subset-eqD node-in-mset-nodes option.distinct(*1*) *option.exhaust-sel option.sel sum-image-mset-sum-map*)
ultimately show *?case*
supply [[*goals-limit=1*]]
using *distinct-mset-union*[*of* ‹∑# (*mset-nodes* ‘# *mset xs*)› ‹∑# (*mset-nodes* ‘# *mset c*)›, *unfolded add.commute*[*of* ‹∑# (*mset-nodes* ‘# *mset xs*)›]]
distinct-mset-union[*of* ‹∑# (*mset-nodes* ‘# *mset c*)› ‹∑# (*mset-nodes* ‘# *mset xs*)›] *2*
apply (*simp-all add*: *remove-key-children-hd-tl split*: *option.splits if-splits*)
apply (*intro conjI impI allI*)
subgoal
apply (*auto split*: *option.splits if-splits simp*: *remove-key-children-hd-tl*)
done
subgoal
apply (*auto split*: *option.splits if-splits simp*: *remove-key-children-hd-tl*)
by (*metis* (*mono-tags*, *lifting*) *fun-comp-eq-conv hp-prev-children.simps*(*2*) *hp-prev-children.simps*(*3*) *hp-prev-children-None-notin hp-prev-children-simps*(*3*) *hp-prev-simps list.collapse sum-image-mset-sum-map*)
subgoal
apply (*auto split*: *option.splits if-splits simp*: *remove-key-children-hd-tl*)
by (*smt* (*verit*, *ccfv-threshold*) *None-eq-map-option-iff* ‹*distinct-mset* (*sum-list* (*map mset-nodes* (*Hp x n* (*remove-key-children k c*) # *remove-key-children k xs*)))› *distinct-mset-add hp-next-children-in-nodes2 hp-prev-children-None-notin hp-prev-in-first-child hp-prev-simps in-find-key-children-notin-remove-key in-the-default-emp*
inter-iff list.map(*2*) *option.exhaust-sel option.map-sel remove-key-children-notin-unchanged sum-image-mset-sum-map sum-list-simps*(*2*) *union-commute union-iff*)
subgoal
apply (*auto split*: *option.splits if-splits simp*: *remove-key-children-hd-tl*)
apply (*simp add*: *hp-prev-children-cons-if*)
apply (*intro conjI impI*)

apply (*metis* (*no-types*, *lifting*) *remove-key-children.simps*(*1*) *WB-List-More.distinct-mset-mono add-diff-cancel-left′ distinct-mset-in-diff hd-remove-key-node-same*
hp.exhaust-sel hp-next-children-in-nodes2 hp-next-children-simps(*2*) *hp-next-children-simps*(*3*) *hp-next-simps list.exhaust-sel mset-nodes-simps*
mset-nodes-find-key-children-subset option.sel option-last-Nil option-last-Some-iff(*2*) *remove-key-children-hd-tl remove-key-remove-all sum-image-mset-sum-map*
union-single-eq-member)
apply (*simp add*: *hp-prev-children-cons-if split*: *option.splits if-splits*)
apply (*metis* ‹*remove-key-children k xs* ≠ [] ⟹ *xs* ≠ []› *hp-next-children-hd-is-hd-tl option-hd-Some-iff*(*2*) *remove-key-children-hd-tl remove-key-children-node-hd sum-image-mset-sum-map*)
by (*metis add-diff-cancel-right′ distinct-mset-in-diff hp-next-children-in-nodes2 hp-prev-children-None-notin option.case-eq-if sum-image-mset-sum-map*)
subgoal
apply (*auto split*: *option.splits if-splits simp*: *remove-key-children-hd-tl*)
apply (*auto simp*: *hp-prev-children-cons-if split*: *option.splits if-splits*)[]
apply (*metis* (*no-types*, *lifting*) *None-eq-map-option-iff in-find-key-children-notin-remove-key in-the-default-empty-iff node-hd-in-sum option.exhaust-sel option.map-sel sum-image-mset-sum-map*)
apply (*metis* (*no-types*, *lifting*) *distinct-mset-add hp-prev-children-None-notin in-the-default-empty-iff inter-iff map-option-is-None mset-map mset-nodes-find-key-children-subset mset-subset-eqD option.map-sel option-last-Nil option-last-Some-iff*(*1*) *sum-mset-sum-list the-default.simps*(*2*))
by (*metis* (*no-types*, *lifting*) *None-eq-map-option-iff add-diff-cancel-right′ distinct-mset-in-diff hp-prev-children-None-notin in-the-default-empty-iff mset-nodes-find-key-children-subset mset-subset-eqD option.map-sel option-last-Nil option-last-Some-iff*(*2*) *sum-image-mset-sum-map*)+
subgoal
apply (*auto split*: *option.splits if-splits simp*: *remove-key-children-empty-iff remove-key-children-hd-tl*)
apply (*auto simp*: *hp-prev-children-cons-if hd-remove-key-node-same′ remove-key-children-empty-iff comp-def split*: *option.splits if-splits*)[]
apply (*metis list.set-sel*(*1*) *node-in-mset-nodes*)
apply (*smt* (*verit*, *best*) *Nil-is-map-conv remove-key-children-alt-def filter-empty-conv hd-in-set hd-remove-key-node-same′ node-in-mset-nodes option.map*(*2*) *the-default.simps*(*1*))
apply (*metis hd-remove-key-node-same hp-next-children-hd-is-hd-tl option-hd-Some-iff*(*2*) *remove-key-children-empty-iff remove-key-children-hd-tl remove-key-children-node-hd sum-image-mset-sum-map*)
apply (*metis* ‹*xs* ≠ [] ⟹ *hp-prev-children* (*node* (*hd xs*)) (*remove-key-children k c*) = *None*› *option.distinct*(*1*) *remove-key-children-notin-unchanged*)
by (*metis* ‹*remove-key-children k xs* ≠ [] ⟹ *hp-prev-children* (*node* (*hd* (*remove-key-children k xs*))) (*remove-key-children k c*) = *None*› *option.distinct*(*1*) *remove-key-children-empty-iff remove-key-children-notin-unchanged*)
subgoal
by (*auto split*: *option.splits if-splits simp*: *remove-key-children-hd-tl*)
subgoal
by (*auto split*: *option.splits if-splits simp*: *remove-key-children-hd-tl*)
subgoal
using *find-None-not-other*
by (*auto split*: *option.splits if-splits simp*: *remove-key-children-hd-tl*)
subgoal
using *find-None-not-other find-key-noneD*[*of k c*]
by (*auto split*: *option.splits if-splits simp*: *remove-key-children-hd-tl*)
subgoal
using *find-None-not-other*
apply (*cases* ‹*k* ∈# *sum-list* (*map mset-nodes c*)›)
apply (*auto split*: *option.splits if-split simp*: *comp-def remove-key-children-hd-tl*)
apply (*auto simp*: *hp-prev-children-cons-if dest*: *mset-nodes-find-key-children-subset split*: *option.splits if-splits*)[]
apply (*metis mset-map mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff*(*1*) *sum-mset-sum-list*)
done
subgoal

using *find-None-not-other*
apply (*auto split*: *option.splits if-splits*
simp: *hp-prev-children-cons-if comp-def remove-key-children-hd-tl*)
done
done
qed

lemma *hp-prev-remove-key-other*:
assumes ‹*distinct-mset* (*mset-nodes xs*)› ‹*remove-key a xs* ≠ *None*›
shows ‹*hp-prev b* (*the* (*remove-key a xs*)) =
(*if b* ∈# (*the-default* {#} (*map-option mset-nodes* (*find-key a xs*))) *then None*
else if map-option node (*hp-next a xs*) = *Some b then* (*hp-prev a xs*)
else map-option (*the o remove-key a*) (*hp-prev b xs*))›
using *assms hp-prev-children-remove-key-children-other*[*of* ‹*hps xs*› *b a*]
by (*cases xs*) *auto*

lemma *hp-next-find-key-children*:
‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset h*)) ⟹ *find-key-children a h* ≠ *None* ⟹
x ∈# *mset-nodes* (*the* (*find-key-children a h*)) ⟹ *x* ≠*a* ⟹
hp-next x (*the* (*find-key-children a h*)) = *hp-next-children x h*›
apply (*induction a h arbitrary*: *x rule*: *find-key-children.induct*)
subgoal
by *auto*
subgoal for *k xa n c xs y*
apply (*auto simp*: *split*: *option.splits*)
apply (*metis add-diff-cancel-left′ distinct-mem-diff-mset hp-next-children-append2*)
apply (*metis disjunct-not-in distinct-mset-add find-key-noneD find-key-none-iff hp.sel*(*1*)
hp-next-None-notin-children hp-next-children-simps(*3*) *mset-map mset-nodes-find-key-children-subset*
mset-subset-eqD option.sel sum-mset-sum-list)
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ distinct-mset-add distinct-mset-in-diff*
find-key-noneD find-key-none-iff hp-next-children-append2 mset-nodes-find-key-children-subset
mset-subset-eqD option.sel sum-image-mset-sum-map)
done

lemma *hp-next-find-key*:
‹*distinct-mset* (*mset-nodes h*) ⟹ *find-key a h* ≠ *None* ⟹ *x* ∈# *mset-nodes* (*the* (*find-key a h*)) ⟹
x ≠*a* ⟹
hp-next x (*the* (*find-key a h*)) = *hp-next x h*›
using *hp-next-find-key-children*[*of* ‹*hps h*› *a x*]
by (*cases* ‹(*a*,*h*)› *rule*: *find-key.cases*;
cases ‹*a* ∈# *sum-list* (*map mset-nodes* (*hps h*))›)
clarsimp-all

lemma *hp-next-find-key-itself*:
‹*distinct-mset* (*mset-nodes h*) ⟹ (*find-key a h*) ≠ *None* ⟹ *hp-next a* (*the* (*find-key a h*)) = *None*›
using *find-key-None-or-itself*[*of a h*]
apply (*cases* ‹*find-key a h*›)
apply *auto*
by (*metis add-diff-cancel-left′ distinct-mset-add-mset distinct-mset-in-diff distinct-mset-mono′*
hp.exhaust-sel hp-next-None-notin-children mset-nodes-simps mset-nodes-find-key-subset option.sel
option.simps(*2*) *sum-mset-sum-list union-mset-add-mset-left*)

lemma *hp-prev-find-key-children*:
‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset h*)) ⟹ *find-key-children a h* ≠ *None* ⟹
x ∈# *mset-nodes* (*the* (*find-key-children a h*)) ⟹ *x* ≠*a* ⟹

hp-prev x (the (find-key-children a h)) = hp-prev-children x h›
apply (induction a h arbitrary: x rule: find-key-children.induct)
subgoal
by auto
subgoal for k xa n c xs y
apply (auto simp: split: option.splits)
apply (simp add: disjunct-not-in distinct-mset-add)
apply (smt (verit, ccfv-SIG) find-key-children.elims remove-key-children.simps(2) WB-List-More.distinct-mset-union2
add-diff-cancel-right′ distinct-mem-diff-mset find-key-noneD hp.sel(1) hp-prev-None-notin-children hp-prev-children-simps
in-find-key-children-notin-remove-key list.distinct(2) list.sel(1) mset-nodes-find-key-children-subset mset-subset-eqD
option.sel option-last-Nil option-last-Some-iff(2) sum-image-mset-sum-map)
by (metis (no-types, lifting) disjunct-not-in distinct-mset-add find-key-noneD find-key-none-iff hp-prev-children-first-chi
mset-nodes-find-key-children-subset mset-subset-eqD option.sel sum-image-mset-sum-map)
done

lemma hp-prev-find-key:
‹distinct-mset (mset-nodes h) ⟹ find-key a h ≠ None ⟹ x ∈# mset-nodes (the (find-key a h)) ⟹
x ≠a ⟹
hp-prev x (the (find-key a h)) = hp-prev x h›
using hp-prev-find-key-children[of ‹hps h› a x]
by (cases ‹(a,h)› rule: find-key.cases;
cases ‹a ∈# sum-list (map mset-nodes (hps h))›)
clarsimp-all

lemma hp-prev-find-key-itself:
‹distinct-mset (mset-nodes h) ⟹ (find-key a h) ≠ None ⟹ hp-prev a (the (find-key a h)) = None›
using find-key-None-or-itself[of a h]
apply (cases ‹find-key a h›)
apply auto
by (metis add-diff-cancel-left′ distinct-mset-add-mset distinct-mset-in-diff distinct-mset-mono′
hp.exhaust-sel hp-prev-None-notin-children mset-nodes-simps mset-nodes-find-key-subset option.sel
option.simps(2) sum-mset-sum-list union-mset-add-mset-left)

lemma hp-child-find-key-children:
‹distinct-mset (∑# (mset-nodes '# mset h)) ⟹ find-key-children a h ≠ None ⟹
x ∈# mset-nodes (the (find-key-children a h)) ⟹
hp-child x (the (find-key-children a h)) = hp-child-children x h›
apply (induction a h arbitrary: x rule: find-key-children.induct)
subgoal
by auto
subgoal for k xa n c xs y
apply (auto simp: hp-child-children-Cons-if split: option.splits)
using distinct-mem-diff-mset **apply** fastforce
apply (metis Groups.add-ac(2) distinct-mset-union find-key-none-iff option.simps(2) sum-image-mset-sum-map)
apply (metis disjunct-not-in distinct-mset-add hp-child-None-notin-children if-Some-None-eq-None
mset-map mset-nodes-find-key-children-subset mset-subset-eqD option.sel sum-mset-sum-list)
apply (metis distinct-mset-union find-key-noneD hp-child-children-None-notin hp-child-children-skip-first
hp-child-children-skip-last
hp-child-hp-children-simps2 mset-map mset-subset-eqD option.sel find-key-none-iff mset-nodes-find-key-children-subse
sum-mset-sum-list)
by (metis (no-types, lifting) distinct-mset-add find-key-noneD find-key-none-iff hp-child-hp-children-simps2
mset-nodes-find-key-children-subset mset-subset-eqD option.sel sum-image-mset-sum-map)
done

lemma hp-child-find-key:
‹distinct-mset (mset-nodes h) ⟹ find-key a h ≠ None ⟹ x ∈# mset-nodes (the (find-key a h)) ⟹

hp-child x (*the* (*find-key a h*)) = *hp-child x h*›
using *hp-child-find-key-children*[*of* ‹*hps h*› *a x*]
apply (*cases* ‹(*a*,*h*)› *rule*: *find-key.cases*;
cases ‹*a* ∈# *sum-list* (*map mset-nodes* (*hps h*))›)
apply *clarsimp-all*
by (*metis find-key-none-iff hp-child-hp-children-simps2 mset-nodes-find-key-children-subset mset-subset-eqD option.sel sum-image-mset-sum-map*)+

lemma *find-remove-children-mset-nodes-full*:
‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *mset h*)) ⟹ *find-key-children a h* = *Some x* ⟹
($\sum_{\#}$ (*mset-nodes* ‘# *mset* (*remove-key-children a h*))) + *mset-nodes x* = $\sum_{\#}$ (*mset-nodes* ‘# *mset h*)›
apply (*induction a h rule*: *find-key-children.induct*)
apply (*auto split*: *if-splits option.splits*)
using *distinct-mset-add* **apply** *blast*
by (*metis* (*no-types*, *lifting*) *disjunct-not-in distinct-mset-add find-key-noneD remove-key-children-notin-unchanged sum-image-mset-sum-map union-assoc union-commute*)

lemma *find-remove-mset-nodes-full*:
‹*distinct-mset* (*mset-nodes h*) ⟹ *remove-key a h* = *Some y* ⟹
find-key a h = *Some ya* ⟹ (*mset-nodes y* + *mset-nodes ya*) = *mset-nodes h*›
apply (*induction a h rule*: *remove-key.induct*)
subgoal for *k x n c*
using *find-remove-children-mset-nodes-full*[*of c k ya*]
by (*auto split*: *if-splits*)
done

lemma *in-remove-key-in-nodes*: ‹*remove-key a h* ≠ *None* ⟹ *x'* ∈# *mset-nodes* (*the* (*remove-key a h*)) ⟹ *x'* ∈# *mset-nodes h*›
by (*metis remove-key.simps hp.exhaust-sel mset-nodes.simps not-orig-notin-remove-key option.sel sum-image-mset-sum-union-iff*)

lemma *in-find-key-in-nodes*: ‹*find-key a h* ≠ *None* ⟹ *x'* ∈# *mset-nodes* (*the* (*find-key a h*)) ⟹ *x'* ∈# *mset-nodes h*›
by (*meson mset-nodes-find-key-subset mset-subset-eqD*)

lemma *in-find-key-notin-remove-key-children*:
‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *mset h*)) ⟹ *find-key-children a h* ≠ *None* ⟹ *x* ∈# *mset-nodes* (*the* (*find-key-children a h*))⟹ *x* ∉# $\sum_{\#}$ (*mset-nodes* ‘# *mset* (*remove-key-children a h*))›
apply (*induction a h rule*: *find-key-children.induct*)
subgoal
by (*auto split*: *if-splits*)
subgoal for *k xa n c xs*
using *distinct-mset-union*[*of* ‹*sum-list* (*map mset-nodes c*)› ‹*sum-list* (*map mset-nodes xs*)›]
distinct-mset-union[*of* ‹*sum-list* (*map mset-nodes c*)› ‹*sum-list* (*map mset-nodes xs*)›]

apply (*auto 4 3 simp*: *remove-key-remove-all*[*simplified*] *ac-simps dest*: *find-key-noneD multi-member-split split*: *option.splits*)
apply (*metis find-key-noneD find-key-none-iff mset-nodes-find-key-children-subset mset-subset-eqD option.sel sum-image-mset-sum-map*)
apply (*metis add-diff-cancel-left' distinct-mset-in-diff mset-nodes-find-key-children-subset mset-subset-eqD option.distinct*(*1*) *option.sel sum-image-mset-sum-map*)
apply (*metis distinct-mset-union find-key-none-iff option.distinct*(*1*) *sum-image-mset-sum-map union-commute*)
apply (*metis mset-nodes-find-key-children-subset mset-subset-eqD option.sel option.simps*(*2*) *sum-image-mset-sum-ma*
apply (*metis find-key-none-iff option.simps*(*2*) *sum-image-mset-sum-map*)
by (*metis add-diff-cancel-right' distinct-mset-in-diff mset-nodes-find-key-children-subset mset-subset-eqD*

not-orig-notin-remove-key option.sel option.simps(*2*) *sum-image-mset-sum-map*)
done

lemma *in-find-key-notin-remove-key*:
‹*distinct-mset* (*mset-nodes h*) ⟹ *find-key a h* ≠ *None* ⟹ *remove-key a h* ≠ *None* ⟹ *x* ∈# *mset-nodes* (*the* (*find-key a h*))⟹ *x* ∉# *mset-nodes* (*the* (*remove-key a h*))›
using *in-find-key-notin-remove-key-children*[*of* ‹*hps h*› *a x*]
apply (*cases h*)
apply *auto*
apply (*metis mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff* (*2*) *sum-image-mset-sum-map*)
by *fastforce*

lemma *map-option-node-hp-next-remove-key*:
‹*distinct-mset* (*mset-nodes h*) ⟹ *map-option node* (*hp-prev a h*) ≠ *Some x′* ⟹ *map-option node* (*hp-next x′ h*) =
map-option (λ*x*. *node* (*the* (*remove-key a x*))) (*hp-next x′ h*)›
apply (*induction x′ h rule*:*hp-next.induct*)
subgoal for *aa m s x y children*
apply (*auto simp*: *split*: *option.splits*)
by (*smt* (*z3*) *remove-key.simps add-mset-add-single distinct-mset-add-mset distinct-mset-union hp.exhaust-sel hp.sel*(*1*) *hp-next-children-simps*(*1*−*3*)
hp-prev-None-notin-children hp-prev-children-None-notin hp-prev-children-simps(*1*) *hp-prev-in-first-child hp-prev-skip-hd-children list.map*(*2*) *list.sel*(*1*)
map-option-cong member-add-mset mset-nodes.simps option.sel option-last-Nil option-last-Some-iff (*2*) *sum-image-mset-sum-map sum-list-simps*(*2*) *union-ac*(*2*))
subgoal by *auto*
subgoal by *auto*
done

lemma *has-prev-still-in-remove-key*: ‹*distinct-mset* (*mset-nodes h*) ⟹ *hp-prev a h* ≠ *None* ⟹
remove-key a h ≠ *None* ⟹ *node* (*the* (*hp-prev a h*)) ∈# *mset-nodes* (*the* (*remove-key a h*))›
apply (*induction a h rule*: *hp-prev.induct*)
subgoal for *a m s x y children*
apply (*cases x*)
apply (*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits*)
using *Duplicate-Free-Multiset.distinct-mset-union2* **apply** *blast*
using *distinct-mset-add* **by** *blast*
subgoal apply *auto*
by (*smt* (*verit*, *del-insts*) *remove-key.simps remove-key-children.elims*
add-diff-cancel-left′ distinct-mset-add-mset hp-prev-children-None-notin hp-prev-simps
insert-DiffM list.distinct(*2*) *list.sel*(*1*) *list.simps*(*9*) *mset-nodes.simps option.sel*
option-last-Nil option-last-Some-iff (*2*) *sum-list-simps*(*2*) *union-iff*)
subgoal by *auto*
done
lemma *find-key-head-node-iff*: ‹*node h* = *node m′* ⟹ *find-key* (*node m′*) *h* = *Some m′* ⟷ *h* = *m′*›
by (*cases h*) *auto*

lemma *map-option-node-hp-prev-remove-key*:
‹*distinct-mset* (*mset-nodes h*) ⟹ *map-option node* (*hp-next a h*) ≠ *Some x′* ⟹ *map-option node* (*hp-prev x′ h*) =
map-option (λ*x*. *node* (*the* (*remove-key a x*))) (*hp-prev x′ h*)›
apply (*induction x′ h rule*:*hp-prev.induct*)
subgoal for *aa m s x y children*
apply (*auto simp*: *hp-prev-children.simps*(*1*) *split*: *option.splits*)
apply (*metis remove-key.simps hp.exhaust-sel hp.sel*(*1*) *hp-next-children-simps*(*1*) *option.sel*)

apply (metis add-diff-cancel-right′ distinct-mset-add distinct-mset-in-diff hp-next-None-notin hp-next-children-None-no
hp-next-children-simps(3) node-in-mset-nodes option-last-Nil option-last-Some-iff(2) sum-image-mset-sum-map union-ac(2))
apply (metis WB-List-More.distinct-mset-union2 add-diff-cancel-right′ distinct-mset-in-diff hp-prev-None-notin node-in-mset-nodes option-last-Nil option-last-Some-iff(2))
by (metis distinct-mset-add find-key-head-node-iff hp-next-children-simps(2) hp-next-find-key-itself option.distinct(1) option.sel)
subgoal by auto
subgoal by auto
done

lemma ‹distinct-mset (mset-nodes h) ⟹ node y ∈# mset-nodes h ⟹ find-key (node y) h = Some y⟹
mset-nodes (the (find-key (node y) h)) = mset-nodes y›
apply (induction ‹node y› h rule: find-key.induct)
apply auto
oops

lemma distinct-mset-find-node-next:
‹distinct-mset (mset-nodes h) ⟹ find-key n h = Some y ⟹
distinct-mset (mset-nodes y + (if hp-next n h = None then {#} else (mset-nodes (the (hp-next n h)))))›
apply (induction n h rule: hp-next.induct)
subgoal for a m s x ya children
apply (cases x)
apply (auto simp: hp-next-children.simps(1)
split: if-splits option.splits)
apply (metis distinct-mset-union union-ac(1))
using distinct-mset-add **apply** blast
using distinct-mset-add **apply** blast
using distinct-mset-add **apply** blast
apply (metis (no-types, opaque-lifting) add-diff-cancel-right′ distinct-mset-add distinct-mset-in-diff find-key-noneD hp-next-None-notin hp-next-children-None-notin hp-next-children-simps(3) node-in-mset-nodes option.simps(2) sum-image-mset-sum-map union-lcomm)
using distinct-mset-add **by** blast
subgoal apply (auto simp: hp-next-children.simps(1)
split: if-splits option.splits)
apply (metis (no-types, opaque-lifting) remove-key-children.simps(1) WB-List-More.distinct-mset-mono arith-extra-simps(5) ex-Melem-conv list.simps(9) mset-nodes-find-key-children-subset option.distinct(2) option.sel remove-key-remove-all sum-image-mset-sum-map sum-list-simps(2) union-ac(2))
by (smt (verit, del-insts) find-key.simps find-key-children.elims find-key-children.simps(1) list-tail-coinc option.case-eq-if option.collapse option.discI)
subgoal
by (auto simp: split: if-splits)
done

lemma hp-child-node-itself[simp]: ‹hp-child (node a) a = option-hd (hps a)›
by (cases a; auto)

lemma find-key-children-itself-hd[simp]:
‹find-key-children (node a) [a] = Some a›
by (cases a; auto)

lemma hp-prev-and-next-same-node:
fixes y h :: ‹('b, 'a) hp›
assumes ‹distinct-mset (mset-nodes h)› ‹hp-prev x′ y ≠ None›

‹*node yb = x′*›
‹*hp-next* (*node y*) *h* = *Some yb*›
‹*find-key* (*node y*) *h* = *Some y*›
shows ‹*False*›
proof −
have *x′y*: ‹*x′* ∈# *mset-nodes y*›
by (*metis assms*(*2*) *hp-prev-None-notin*)
have ‹*x′* ≠ *node y*›
using *assms*(*1*,*2*) **by** (*metis assms*(*3*) *assms*(*4*) *hp-next-not-same-node*)
have ‹*remove-key* (*node y*) *h* ≠ *None*›
by (*metis remove-key-None-iff find-key-head-node-iff handy-if-lemma hp-next-find-key-itself option.sel assms*(*1*,*4*))
moreover have *neq*: ‹*find-key* (*node y*) *h* ≠ *None*›
by (*metis find-key.elims find-key-none-iff hp-next-children-None-notin hp-next-simps option.discI assms*(*4*))
ultimately have ‹*node* (*the* (*hp-next* (*node y*) *h*)) ≠ *x′*›
using *hp-next-remove-key-other*[*of h* ‹*node y*› *x′*]
distinct-mset-find-node-next[*of h* ‹*node y*›] *assms*
by (*cases yb*) *auto*
then show *False*
using *assms* **by** (*auto split*: *if-splits simp*:)
qed

lemma *hp-child-children-remove-is-remove-hp-child-children*:
‹*distinct-mset* (∑# (*mset-nodes* ‘# *mset c*)) ⟹
hp-child-children b (*c*) ≠ *None* ⟹
hp-parent-children k (*c*) = *None* ⟹
hp-child-children b ((*remove-key-children k c*)) ≠ *None* ⟹
(*hp-child-children b* (*remove-key-children k c*)) = (*remove-key k* (*the* (*hp-child-children b* (*c*))))›
apply (*induction k c rule*: *remove-key-children.induct*)
subgoal by *auto*
subgoal for *k x n c xs*
using *distinct-mset-union*[*of* ‹*sum-list* (*map mset-nodes c*)› ‹*sum-list* (*map mset-nodes xs*)›]
apply *auto*
apply (*metis disjunct-not-in distinct-mset-add hp-child-None-notin-children hp-child-children-None-notin hp-child-children-simps*(*2*) *option.sel option-last-Nil option-last-Some-iff*(*2*) *sum-image-mset-sum-map*)
apply (*auto simp*: *hp-parent-children-cons split*: *option.splits*)
by (*smt* (*verit*) *remove-key.simps remove-key-children.elims disjunct-set-mset-diff distinct-mset-add distinct-mset-in-diff hp.sel*(*1*) *hp-child-children-Cons-if hp-child-children-None-notin hp-child-hd hp-child-hp-children-simps2*
hp-parent-None-iff-children-None list.sel(*1*) *mset-subset-eqD node-remove-key-children-in-mset-nodes option.sel option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)
done

lemma *hp-child-remove-is-remove-hp-child*:
‹*distinct-mset* (*mset-nodes* (*Hp x n c*)) ⟹
hp-child b (*Hp x n c*) ≠ *None* ⟹
hp-parent k (*Hp x n c*) = *None* ⟹
remove-key k (*Hp x n c*) ≠ *None* ⟹
hp-child b (*the* (*remove-key k* (*Hp x n c*))) ≠ *None* ⟹
hp-child b (*the* (*remove-key k* (*Hp x n c*))) = *remove-key k* (*the* (*hp-child b* (*Hp x n c*)))›
using *hp-child-children-remove-is-remove-hp-child-children*[*of c b k*]
apply *auto*
by (*smt* (*z3*) *remove-key.elims remove-key-children.elims hp.exhaust-sel hp.inject hp-child-hd hp-child-hp-children-simps2 hp-parent-None-iff-children-None list.sel*(*1*) *option.sel option-hd-Some-iff*(*2*))

lemma *remove-key-children-itself-hd*[*simp*]: ‹*distinct-mset* (*mset-nodes a* + *sum-list* (*map mset-nodes list*)) ⟹
remove-key-children (*node a*) (*a* # *list*) = *list*›
by (*cases a*; *auto*)

lemma *hp-child-children-remove-key-children-other-helper*:
assumes
K: ‹*hp-child-children b* (*remove-key-children k c*) = *map-option* ((*the* ∘∘ *remove-key*) *k*) (*hp-child-children b c*)› **and**
H: ‹*node x2a* ≠ *b*›
‹*hp-parent k* (*Hp x n c*) = *Some x2a*›
‹*hp-child b* (*Hp x n c*) = *Some y*›
‹*hp-child b* (*Hp x n* (*remove-key-children k c*)) = *Some ya*›
shows
‹*ya* = *the* (*remove-key k y*)›
using *K H*
apply (*cases c*; *cases y*)
apply (*auto split*: *option.splits if-splits*)
apply (*metis get-min2.simps get-min2-alt-def hp-parent-simps*(*1*))
by (*metis get-min2.simps get-min2-alt-def hp-parent-simps*(*1*))

lemma *hp-child-children-remove-key-children-other*:
assumes ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *mset xs*))›
shows ‹*hp-child-children b* (*remove-key-children a xs*) =
(*if b* ∈# (*the-default* {#} (*map-option mset-nodes* (*find-key-children a xs*))) *then None*
else if map-option node (*hp-parent-children a xs*) = *Some b then* (*hp-next-children a xs*)
else map-option (*the o remove-key a*) (*hp-child-children b xs*))›
using *assms*
proof (*induction a xs rule*: *remove-key-children.induct*)
case (*1 k*)
then show *?case* **by** *auto*
next
case (*2 k x n c xs*)
moreover have ‹*distinct-mset*
(*sum-list* (*map mset-nodes c*) + *sum-list* (*map mset-nodes* (*remove-key-children k xs*)))›
‹*x* ∉# *sum-list* (*map mset-nodes* (*remove-key-children k xs*))›
using *2*(*4*) **apply** *auto*
apply (*metis distinct-mset-mono′ mset-map mset-subset-eq-mono-add-left-cancel node-remove-key-children-in-mset-nodes sum-mset-sum-list*)
by (*simp add*: *not-orig-notin-remove-key*)
moreover have ‹*distinct-mset*
(*sum-list*
(*map mset-nodes* (*Hp x n* (*remove-key-children k c*) # *remove-key-children k xs*)))›
using *2*(*4*) **apply** (*auto simp*: *not-orig-notin-remove-key*)
by (*metis calculation*(*5*) *distinct-mset-mono′ mset-map node-remove-key-children-in-mset-nodes subset-mset.add-right-mono sum-mset-sum-list*)
moreover have ‹*hp-prev-children k xs* ≠ *None* ⟹ *remove-key-children k xs* ≠ []›
using *2*(*4*) **by** (*cases xs*; *cases* ‹*hd xs*›; *cases* ‹*tl xs*›) (*auto*)
moreover have ‹*x* = *node z* ⟹ *hp-prev-children k* (*Hp* (*node z*) *n c* # *xs*) = *Some z* ⟷
z = *Hp x n c* ∧ *xs* ≠ [] ∧ *k* = *node* (*hd* (*xs*))› **for** *z*
using *2*(*4*) *hp-prev-children-in-nodes*[*of* - *c*] **apply** −
apply (*cases* ‹*xs*›; *cases z*; *cases* ‹*hd xs*›)
using *hp-prev-children-in-nodes*[*of* - *c*] **apply** *fastforce*
apply (*auto simp*:)
apply (*metis 2*(*4*) *hp.inject hp.sel*(*1*) *hp-prev-children-in-nodes hp-prev-children-simps*(*1*) *hp-prev-children-simps*(*2*)

hp-prev-children-simps(3) hp-prev-simps list.distinct(1) list.sel(1) list.sel(3) option.sel remove-key-children-hd-tl remove-key-remove-all sum-image-mset-sum-map)
apply (metis 2(4) hp.inject hp.sel(1) hp-prev-children-in-nodes hp-prev-children-simps(1) hp-prev-children-simps(2) hp-prev-children-simps(3) hp-prev-simps in-remove-key-children-changed list.distinct(2) list.sel(1) list.sel(3) option.sel remove-key-children-hd-tl sum-image-mset-sum-map)
by (metis 2(4) hp.sel(1) hp-prev-children-in-nodes hp-prev-children-simps(2) hp-prev-children-simps(3) hp-prev-simps in-remove-key-children-changed list.distinct(2) list.sel(1) list.sel(3) option.sel remove-key-children-hd-tl sum-image-mset-sum-map)
moreover have ‹xs ≠ [] ⟹ find-key-children (node (hd xs)) xs = Some (hd xs)›
by (metis find-key-children.simps(2) hp.exhaust-sel list.exhaust-sel)
ultimately show ?case
using distinct-mset-union[of ‹$\sum_{\#}$ (mset-nodes '# mset xs)› ‹$\sum_{\#}$ (mset-nodes '# mset c)›, unfolded add.commute[of ‹$\sum_{\#}$ (mset-nodes '# mset xs)›]]
distinct-mset-union[of ‹$\sum_{\#}$ (mset-nodes '# mset c)› ‹$\sum_{\#}$ (mset-nodes '# mset xs)›]
apply (auto split: option.splits if-splits simp: remove-key-children-hd-tl in-the-default-empty-iff)
apply (simp add: disjunct-not-in distinct-mset-add)
apply (auto simp add: hp-parent-children-cons hp-child-children-Cons-if)[]
apply (metis disjunct-not-in distinct-mset-add hp-child-children-None-notin hp-child-hp-children-simps2 hp-parent-children-in-nodes2 option.distinct(1) sum-image-mset-sum-map)
apply (metis add-diff-cancel-left′ distinct-mset-in-diff hp-child-None-notin-children hp-child-children-simps(2) hp-parent-children-in-nodes2 sum-image-mset-sum-map)
apply (simp add: hp-parent-children-cons)
apply (simp add: hp-child-children-Cons-if)
apply (metis disjunct-not-in distinct-mset-add find-key-none-iff hp-child-None-notin-children mset-map mset-nodes-find-key-children-subset mset-subset-eqD option.sel sum-mset-sum-list)
apply (smt (verit, del-insts) hp-child.simps(2) hp-child-children-Cons-if hp-child-hd hp-child-hp-children-simps2 list.exhaust-sel list.simps(9) o-apply option.map(2) option.sel option-hd-Some-hd remove-key-notin-unchanged sum-list-simps(2) union-iff)
apply (metis hp-parent-None-iff-children-None hp-parent-children-None-notin hp-parent-children-append-case hp-parent-children-append-skip-first hp-parent-children-cons mset-map node-hd-in-sum sum-mset-sum-list)

apply (auto simp add: hp-child-children-Cons-if)[]
apply (smt (verit, best) ex-hp-node-children-Some-in-mset-nodes hp.sel(1) hp-child-children-remove-is-remove-hp-chil hp-child-hd hp-child-hp-children-simps2 hp-node-children-None-notin2 hp-parent-children-remove-key-children list.sel(1) option.sel option-hd-Some-iff(2) hd-remove-key-node-same remove-key.simps remove-key-children.elims remove-key-children-notin-unchanged sum-image-mset-sum-map)
apply (metis add-diff-cancel-left′ distinct-mset-in-diff)
apply (metis add-diff-cancel-left′ distinct-mset-in-diff hp-parent-children-None-notin option-last-Nil option-last-Some-iff(2))
apply (metis (mono-tags, lifting) add-diff-cancel-right′ distinct-mset-in-diff hp-child-None-notin-children hp-child-children-None-notin hp-child-children-simps(2) in-find-key-children-notin-remove-key mset-nodes-find-key-childre mset-subset-eqD option.sel option-last-Nil option-last-Some-iff(2) sum-image-mset-sum-map)
apply (smt (verit, del-insts) arith-simps(49) disjunct-not-in distinct-mset-add hp-child-None-notin hp-child-children-None-notin hp-child-children-simps(2) in-find-key-notin-remove-key-children member-add-mset mset-nodes-find-key-children-subset mset-nodes-simps option.distinct(1) option.sel subset-mset.le-iff-add sum-image-mset-sum-map union-iff)
apply (metis add-diff-cancel-right′ distinct-mset-in-diff find-key-noneD sum-image-mset-sum-map)
apply (metis disjunct-not-in distinct-mset-add hp-parent-children-in-nodes2 sum-image-mset-sum-map)
apply (metis add-diff-cancel-right′ distinct-mset-in-diff hp-child-children-Cons-if hp-child-children-None-notin hp-child-hp-children-simps2 hp-parent-children-in-nodes2 sum-image-mset-sum-map)
apply (auto simp add: hp-child-children-Cons-if hp-parent-children-in-nodes2)[]
apply (metis disjunct-not-in distinct-mset-add find-key-noneD hp-child-children-None-notin hp-child-hp-children-simp hp-next-children-append2 hp-parent-children-in-nodes2 mset-map sum-mset-sum-list)
apply (metis hp.sel(1) hp-child-hp-children-simps2 hp-next-children-simps(2) hp-next-simps hp-parent-children-in-noc option.distinct(1) option.sel sum-image-mset-sum-map)

apply (*metis add-diff-cancel-right′ distinct-mset-in-diff find-key-noneD sum-image-mset-sum-map*)
apply (*metis disjunct-not-in distinct-mset-add find-key-noneD hp-parent-children-None-notin option.distinct*(*1*) *sum-image-mset-sum-map*)
apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*metis get-min2.simps get-min2-alt-def hp-child-children-None-notin hp-child-hd hp-next-children-hd-is-hd-tl hp-parent-simps-single-if option-last-Nil option-last-Some-iff* (*2*) *remove-key-children-hd-tl remove-key-children-node-hd sum-image-mset-sum-map*)
apply (*metis get-min2.simps get-min2-alt-def hp-child-hd hp-next-children-hd-is-hd-tl hp-parent-simps-single-if option-last-Nil option-last-Some-iff* (*2*) *remove-key-children-hd-tl remove-key-children-node-hd sum-image-mset-sum-map*)

apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*smt* (*z3*) *add-diff-cancel-right′ distinct-mset-in-diff find-key-children-notin get-min2.simps get-min2-alt-def hp.sel*(*3*) *hp-child.elims hp-child-children-None-notin hp-next-children-append2 hp-next-children-hd-is-hd-hp-parent-simps-single-if option-hd-Nil option-last-Nil option-last-Some-iff* (*2*) *remove-key-children-hd-tl remove-key-children-node-hd sum-image-mset-sum-map*)
apply (*metis get-min2.simps get-min2-alt-def hp-child-hd hp-next-children-hd-is-hd-tl hp-next-children-simps*(*2*) *hp-next-simps hp-parent-simps-single-if option-last-Nil option-last-Some-iff* (*2*) *remove-key-children-hd-tl remove-key-children-node-hd sum-image-mset-sum-map*)

apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*smt* (*z3*) *add-diff-cancel-right′ distinct-mset-in-diff find-key-children-notin get-min2.simps get-min2-alt-def hp.sel*(*3*) *hp-child.elims hp-child-children-None-notin hp-next-children-append2 hp-next-children-hd-is-hd-hp-parent-simps-single-if option-hd-Nil option-last-Nil option-last-Some-iff* (*2*) *remove-key-children-hd-tl remove-key-children-node-hd sum-image-mset-sum-map*)
apply (*meson disjunct-not-in distinct-mset-add*)

apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*metis disjunct-not-in distinct-mset-add hp-next-children-None-notin sum-image-mset-sum-map*)
apply (*metis hp-child-hp-children-simps2 hp-parent-children-in-nodes option.distinct*(*1*) *option.sel sum-image-mset-sum-map*)

apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ distinct-mset-in-diff hp-child-children-None-notin mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff* (*2*) *sum-image-mset-su*
apply (*metis hp-child-hp-children-simps2 mset-nodes-find-key-children-subset mset-subset-eqD option.distinct*(*1*) *option.sel sum-image-mset-sum-map*)
apply (*metis* (*no-types*, *lifting*) *add-diff-cancel-left′ distinct-mset-in-diff hp-child-children-None-notin mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff* (*2*) *sum-image-mset-su*
apply (*metis hp-child-hp-children-simps2 mset-nodes-find-key-children-subset mset-subset-eqD option.distinct*(*1*) *option.sel sum-image-mset-sum-map*)

apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*metis disjunct-not-in distinct-mset-add hp-child-children-None-notin mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff* (*2*) *sum-image-mset-sum-map*)
apply (*metis hp-child-hp-children-simps2 mset-nodes-find-key-children-subset mset-subset-eqD option.distinct*(*1*) *option.sel sum-image-mset-sum-map*)
apply (*metis disjunct-not-in distinct-mset-add hp-child-children-None-notin mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff* (*2*) *sum-image-mset-sum-map*)
apply (*metis hp-child-hp-children-simps2 mset-nodes-find-key-children-subset mset-subset-eqD option.distinct*(*1*) *option.sel sum-image-mset-sum-map*)

apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)

apply (*metis add-diff-cancel-right′ distinct-mset-in-diff hp-parent-children-None-notin option-last-Nil option-last-Some-iff*(*2*))

apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)

apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis hp-parent-None-iff-children-None option.distinct*(*1*))
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)
apply (*metis get-min2-alt-def hp-parent-children-hd-None hp-parent-simps-single-if option.distinct*(*1*) *sum-image-mset-sum-map*)

apply (*metis add-diff-cancel-right′ distinct-mset-in-diff find-key-noneD sum-image-mset-sum-map*)

apply (*metis disjunct-not-in distinct-mset-add find-key-noneD hp-parent-children-None-notin option.distinct*(*1*) *sum-image-mset-sum-map*)

apply (*auto simp add*: *hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons split*: *option.splits if-splits*)[]
apply (*metis find-key-children.simps*(*1*) *hp-child-hd hp-child-hp-children-simps2 option.distinct*(*1*) *option.simps*(*8*) *option-hd-None-iff*(*2*))
apply (*smt* (*verit*, *best*) *find-key-children.elims find-key-children-None-or-itself find-key-noneD find-key-none-iff hp.inject hp-child-hd hp-child-hp-children-simps2 hp-parent-None-iff-children-None*

list.discI list.sel(1)
map-option-is-None mset-map option.sel option-hd-None-iff(1) remove-key-children.elims sum-mset-sum-list)
apply (*metis (no-types, lifting) remove-key.simps ex-hp-node-children-Some-in-mset-nodes hp-child-remove-is-remove-*
hp-node-children-None-notin2 is-mset-set-add mset-nodes.simps option.sel sum-image-mset-sum-map
union-ac(2))
apply (*metis remove-key-children.simps(1) hp-child.simps(1) hp-child-hp-children-simps2 neq-NilE*
option.distinct(1) option.simps(8))
apply (*smt (verit, ccfv-SIG) remove-key-children.elims find-key-children-None-or-itself find-key-noneD*
find-key-none-iff get-min2.simps
get-min2-alt-def hp.inject hp-child-hd hp-child-hp-children-simps2 hp-parent-simps-single-if list.discI
option.map-disc-iff option-hd-None-iff(2)
option-last-Nil option-last-Some-iff(1) remove-key-children-hp-parent-children-hd-None remove-key-children-notin-u
apply (*meson hp-child-children-remove-key-children-other-helper*)

apply (*auto simp add: hp-child-children-Cons-if hp-parent-children-in-nodes2 hp-parent-children-cons*
split: option.splits if-splits)[]
apply (*metis find-key-children.simps(1) hp-child-hd hp-child-hp-children-simps2 option.distinct(1)*
option.simps(8) option-hd-None-iff(2))
apply (*smt (verit) find-key-children-None-or-itself hp.inject hp-child-hd hp-child-hp-children-simps2*
hp-parent-simps(1) list-tail-coinc map-option-is-None neq-Nil-conv not-Some-eq option.sel option-hd-None-iff(1)
find-key-children.elims remove-key-children.elims)

apply (*metis (no-types, lifting) remove-key.simps ex-hp-node-children-Some-in-mset-nodes hp-child-remove-is-remove-*
hp-node-children-None-notin2 is-mset-set-add mset-nodes.simps option.sel sum-image-mset-sum-map union-ac(2))
apply (*metis find-key-children.simps(1) hp-child-hd hp-child-hp-children-simps2 option.distinct(1)*
option.simps(8) option-hd-None-iff(2))
apply (*smt (verit) find-key-children.elims find-key-children-None-or-itself find-key-noneD find-key-none-iff*
get-min2.simps get-min2-alt-def hp.inject hp-child-hd hp-child-hp-children-simps2 hp-parent-simps-single-if
if-Some-None-eq-None list-tail-coinc map-option-is-None neq-Nil-conv option.sel option-hd-None-iff(1)
find-key-children.elims remove-key-children.elims remove-key-children-hp-parent-children-hd-None remove-key-children-n
by (*meson hp-child-children-remove-key-children-other-helper*)
qed

lemma *hp-child-remove-key-other*:
assumes ‹*distinct-mset (mset-nodes xs)*› ‹*remove-key a xs ≠ None*›
shows ‹*hp-child b (the (remove-key a xs)) =*
(if b ∈# (the-default {#} (map-option mset-nodes (find-key a xs))) then None
else if map-option node (hp-parent a xs) = Some b then (hp-next a xs)
else map-option (the o remove-key a) (hp-child b xs))›
using *assms hp-child-children-remove-key-children-other[of ‹hps xs› b a]*
apply (*cases xs*)
apply *auto*
apply (*metis hp-child-hp-children-simps2 in-the-default-empty-iff mset-map mset-nodes-find-key-children-subset*
mset-subset-eqD option.map-disc-iff option.map-sel sum-mset-sum-list)
apply (*metis get-min2.simps get-min2-alt-def hp.sel(3) hp-child-hd hp-child-hp-children-simps2 hp-next-children-hd-is-h*
hp-parent-children-in-nodes2 hp-parent-simps-single-if option-last-Nil option-last-Some-iff(2) remove-key-children-hd-tl
remove-key-children-node-hd sum-image-mset-sum-map)
apply (*metis hp-child-hp-children-simps2 in-the-default-empty-iff map-option-is-None mset-map mset-nodes-find-key-chi*
mset-subset-eqD option.map-sel sum-mset-sum-list)
apply (*case-tac x3; case-tac ‹hd x3›*)
apply (*auto simp add: hp-parent-simps-single-if split: option.splits if-splits*)
done

abbreviation *hp-score-children* **where**
‹*hp-score-children a xs ≡ map-option score (hp-node-children a xs)*›

lemma *hp-score-children-remove-key-children-other*:
assumes ‹*distinct-mset* (∑# (*mset-nodes* ‘# *mset xs*))›
shows ‹*hp-score-children b* (*remove-key-children a xs*) =
(*if b* ∈# (*the-default* {#} (*map-option mset-nodes* (*find-key-children a xs*))) *then None*
else (*hp-score-children b xs*))›
using *assms* **apply** (*induction a xs rule*: *remove-key-children.induct*)
apply (*auto split*: *option.splits if-splits simp*: *hp-node-children-Cons-if in-the-default-empty-iff*)
apply (*metis find-key-none-iff mset-nodes-find-key-children-subset mset-subset-eqD option.map-sel sum-image-mset-sum*
apply (*simp add*: *disjunct-not-in distinct-mset-add*)
apply (*metis disjunct-not-in distinct-mset-add find-key-none-iff mset-nodes-find-key-children-subset mset-subset-eqD option.map-sel sum-image-mset-sum-map*)
apply (*metis None-eq-map-option-iff distinct-mset-add find-key-noneD sum-image-mset-sum-map*)
using *distinct-mset-add* **apply** *blast*
apply (*metis mset-nodes-find-key-children-subset mset-subset-eqD option.distinct*(*2*) *option.sel sum-image-mset-sum-ma*
apply (*meson not-orig-notin-remove-key*)
apply (*metis disjunct-not-in distinct-mset-add hp-node-children-None-notin2 not-orig-notin-remove-key sum-image-mset-sum-map*)
apply (*metis distinct-mset-add hp-node-children-None-notin2 sum-image-mset-sum-map*)
apply (*metis Duplicate-Free-Multiset.distinct-mset-union2 None-eq-map-option-iff find-key-noneD hp-node-children-Nor sum-image-mset-sum-map union-commute*)
apply (*metis None-eq-map-option-iff distinct-mset-add hp-node-children-None-notin2 sum-image-mset-sum-map*)
apply (*metis mset-nodes-find-key-children-subset mset-subset-eqD option.distinct*(*2*) *option.sel sum-image-mset-sum-ma*
apply (*metis add-diff-cancel-right′ distinct-mset-add distinct-mset-in-diff find-key-noneD sum-image-mset-sum-map*)
apply (*meson not-orig-notin-remove-key*)
by (*meson not-orig-notin-remove-key*)

abbreviation *hp-score* **where**
‹*hp-score a xs* ≡ *map-option score* (*hp-node a xs*)›

lemma *hp-score-remove-key-other*:
assumes ‹*distinct-mset* (*mset-nodes xs*)› ‹*remove-key a xs* ≠ *None*›
shows ‹*hp-score b* (*the* (*remove-key a xs*)) =
(*if b* ∈# (*the-default* {#} (*map-option mset-nodes* (*find-key a xs*))) *then None*
else (*hp-score b xs*))›
using *hp-score-children-remove-key-children-other*[*of* ‹*hps xs*› *b a*] *assms*
apply (*cases xs*)
apply (*auto split*: *if-splits simp*: *in-the-default-empty-iff*)
apply (*metis mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff*(*2*) *sum-image-mset-sum-map*)
apply (*simp add*: *hp-node-children-None-notin2*)
by (*metis hp.sel*(*2*) *hp-node-children-simps2 hp-node-simps option.simps*(*9*))

lemma *map-option-node-remove-key-iff*:
‹(*h* ≠ *None* ⟹ *distinct-mset* (*mset-nodes* (*the h*))) ⟹ (*h* ≠ *None* ⟹ *remove-key a* (*the h*) ≠ *None*) ⟹
map-option node h = *map-option node* (*map-option* (*λx. the* (*remove-key a x*)) *h*) ⟷ *h* = *None* ∨ (*h* ≠ *None* ∧ *a* ≠ *node* (*the h*))›
apply (*cases h*; *cases* ‹*the h*›)
apply *simp*
apply (*rule iffI*)
apply *simp*
apply *auto*
done

lemma *sum-next-prev-child-subset*:

‹*distinct-mset* (*mset-nodes h*) ⟹
((*if hp-next n h* = *None then* {#} *else* (*mset-nodes* (*the* (*hp-next n h*)))) +
(*if hp-prev n h* = *None then* {#} *else* (*mset-nodes* (*the* (*hp-prev n h*)))) +
(*if hp-child n h* = *None then* {#} *else* (*mset-nodes* (*the* (*hp-child n h*))))) ⊆# *mset-nodes h*›
apply (*induction n h rule*: *hp-next.induct*)
apply (*auto split*: *option.splits if-splits*)
apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *split*: *if-splits option.splits intro*: *distinct-mset-mono'*)[]
apply (*metis distinct-mset-add mset-le-incr-right*(*2*) *union-mset-add-mset-right*)
apply (*smt* (*verit*, *ccfv-threshold*) *distinct-mset-add hp-next-children-simps*(*1*) *hp-next-children-simps*(*2*) *hp-prev-children-simps*(*1*) *hp-prev-children-simps*(*2*) *hp-prev-children-simps*(*3*) *hp-prev-simps mset-subset-eq-add-right option.sel option-last-Nil option-last-Some-iff* (*2*) *subset-mset.dual-order.trans union-commute union-mset-add-mset-right*
apply (*smt* (*verit*, *ccfv-threshold*) *Duplicate-Free-Multiset.distinct-mset-union2 Groups.add-ac*(*2*) *ab-semigroup-add-clas add-diff-cancel-left' distinct-mem-diff-mset hp-child-children-None-notin hp-next-children-simps*(*2*) *hp-prev-children-simps hp-prev-children-simps*(*3*) *hp-prev-simps mset-le-subtract-right mset-map mset-subset-eq-mono-add node-in-mset-nodes option.sel option-last-Nil option-last-Some-iff* (*1*) *sum-mset-sum-list union-mset-add-mset-right*)
apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *split*: *if-splits option.splits intro*: *distinct-mset-mono'*)[]
apply (*metis distinct-mset-add mset-le-incr-right*(*2*) *union-mset-add-mset-right*)
apply (*metis distinct-mset-add mset-le-incr-right*(*1*) *union-mset-add-mset-right*)

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in*
split: *if-splits option.splits intro*: *distinct-mset-mono'*
intro: *mset-le-incr-right*)[]
apply (*metis mset-le-incr-right*(*2*) *union-mset-add-mset-right*)
apply (*metis hp-child-children-None-notin hp-next-None-notin option.simps*(*2*) *sum-image-mset-sum-map*)
apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in*
split: *if-splits option.splits intro*: *distinct-mset-mono'*
intro: *mset-le-incr-right*)[]
apply (*metis hp-next-None-notin node-in-mset-nodes option.simps*(*2*))
apply (*metis mset-le-incr-right*(*2*) *union-mset-add-mset-right*)
subgoal
by (*metis hp-next-None-notin hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-simps*(*2*) *hp-prev-None-notin-children hp-prev-simps mset-map option.simps*(*2*) *sum-mset-sum-list*)
subgoal
by (*metis hp-prev-None-notin node-in-mset-nodes option-last-Nil option-last-Some-iff* (*2*))
subgoal
by (*metis subset-mset.add-mono union-ac*(*2*) *union-mset-add-mset-right*)
subgoal
by (*metis hp-next-None-notin hp-next-children-simps*(*3*) *hp-next-children-skip-end hp-prev-None-notin option-hd-Nil option-hd-Some-iff* (*2*))
subgoal
by (*metis mset-le-incr-right*(*1*) *union-mset-add-mset-right*)

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in*
split: *if-splits option.splits intro*: *distinct-mset-mono'*
intro: *mset-le-incr-right*)[]
subgoal
by (*metis mset-le-incr-right*(*2*) *union-mset-add-mset-right*)
subgoal
by (*metis hp-child-children-None-notin hp-next-None-notin option-hd-Nil option-hd-Some-iff* (*2*) *sum-image-mset-sum-map*)
subgoal

by (*metis hp-child-children-None-notin hp-prev-None-notin option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)
subgoal
by (*metis hp-child-children-None-notin hp-prev-None-notin option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)
subgoal
by (*metis hp-child-children-None-notin hp-next-None-notin mset-map option.simps*(*2*) *sum-mset-sum-list*)

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in*
split: *if-splits option.splits intro*: *distinct-mset-mono′*
intro: *mset-le-incr-right mset-le-incr-right subset-mset-imp-subset-add-mset*)[]

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in*
split: *if-splits option.splits intro*: *distinct-mset-mono′*
intro: *mset-le-incr-right mset-le-incr-right subset-mset-imp-subset-add-mset*)[]
subgoal
by (*metis hp-child-None-notin node-in-mset-nodes option-last-Nil option-last-Some-iff*(*2*))
subgoal
by (*metis subset-mset.add-mono union-ac*(*2*) *union-mset-add-mset-right*)
subgoal
by (*metis hp-child-None-notin hp-child-children-None-notin option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)
subgoal
by (*metis hp-child-None-notin node-in-mset-nodes option-last-Nil option-last-Some-iff*(*2*))

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in*
split: *if-splits option.splits intro*: *distinct-mset-mono′*
intro: *mset-le-incr-right mset-le-incr-right subset-mset-imp-subset-add-mset*)[]
subgoal
by (*smt* (*verit, del-insts*) *hp.collapse list.exhaust-sel list.simps*(*9*) *mset-le-decr-left*(*1*) *mset-map mset-nodes.simps subset-mset.dual-order.refl subset-mset-trans-add-mset sum-list-simps*(*2*) *sum-mset-sum-list union-ac*(*2*))
subgoal
by (*metis subset-mset.add-mono union-ac*(*2*) *union-mset-add-mset-right*)
subgoal
by (*metis hp-child-None-notin hp-child-children-None-notin option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in*
split: *if-splits option.splits intro*: *distinct-mset-mono′*
intro: *mset-le-incr-right mset-le-incr-right subset-mset-imp-subset-add-mset*)[]
subgoal
by (*metis node-in-mset-nodes*)
subgoal
by (*metis hp-child-None-notin node-in-mset-nodes option-last-Nil option-last-Some-iff*(*2*))
subgoal
by (*metis hp-child-None-notin node-in-mset-nodes option-last-Nil option-last-Some-iff*(*2*))

subgoal
by (*metis subset-mset.add-mono union-commute union-mset-add-mset-right*)
subgoal
by (*metis hp-child-None-notin hp-child-children-None-notin option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)
subgoal
by (*metis hp-child-None-notin hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-simps*(*2*) *hp-prev-children-None-notin mset-map option.distinct*(*1*) *sum-mset-sum-list*)
subgoal
by (*metis hp-prev-None-notin node-in-mset-nodes option-last-Nil option-last-Some-iff*(*2*))
subgoal
by (*metis hp-prev-None-notin node-in-mset-nodes option-last-Nil option-last-Some-iff*(*2*))
subgoal
by (*metis hp-child-None-notin hp-next-None-notin hp-next-children-None-notin hp-next-children-simps*(*3*) *option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)
subgoal
by (*metis hp-next-None-notin hp-next-children-None-notin hp-next-children-simps*(*3*) *hp-prev-None-notin option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)
subgoal
by (*metis hp-child-children-None-notin hp-prev-None-notin option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)
subgoal
by (*metis hp-child-None-notin hp-child-children-None-notin option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map*)

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in ac-simps*
split: *if-splits option.splits intro*: *distinct-mset-mono*′
intro: *mset-le-incr-right mset-le-incr-right subset-mset-imp-subset-add-mset*)[]
apply (*metis mset-le-incr-right2 union-mset-add-mset-right*)
apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in ac-simps*
split: *if-splits option.splits intro*: *distinct-mset-mono*′
intro: *mset-le-incr-right mset-le-incr-right subset-mset-imp-subset-add-mset*)[]
subgoal
by (*metis mset-le-incr-right2 union-mset-add-mset-right*)
subgoal
by (*metis hp-child-None-notin hp-prev-None-notin option.distinct*(*1*))
subgoal
by (*metis hp-child-None-notin hp-prev-None-notin option.distinct*(*1*))

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in ac-simps*
split: *if-splits option.splits intro*: *distinct-mset-mono*′
intro: *mset-le-incr-right mset-le-incr-right subset-mset-imp-subset-add-mset*)[]
subgoal
by (*metis mset-le-incr-right2 union-mset-add-mset-right*)
subgoal
by (*metis hp-child-None-notin hp-next-None-notin option.distinct*(*1*))

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in ac-simps*
split: *if-splits option.splits intro*: *distinct-mset-mono*′ *union-mset-add-mset-right*

intro: *mset-le-incr-right mset-le-incr-right2 subset-mset-imp-subset-add-mset*)[]
subgoal
by (*metis hp-next-None-notin node-in-mset-nodes option-last-Nil option-last-Some-iff*(*2*))
subgoal
by (*metis mset-le-incr-right2 union-mset-add-mset-right*)
subgoal
by (*metis hp-child-None-notin hp-next-None-notin option-hd-Nil option-hd-Some-iff*(*2*))
subgoal
by (*metis hp-next-None-notin node-in-mset-nodes option.simps*(*2*))
subgoal
by (*metis hp-child-None-notin hp-prev-None-notin option-hd-Nil option-hd-Some-iff*(*2*))
subgoal
by (*metis hp-child-None-notin hp-prev-None-notin option-hd-Nil option-hd-Some-iff*(*2*))
subgoal
by (*metis hp-child-None-notin hp-next-None-notin option.simps*(*2*))

apply (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in ac-simps*
split: *if-splits option.splits intro*: *distinct-mset-mono′ union-mset-add-mset-right*
intro: *mset-le-incr-right mset-le-incr-right2 subset-mset-imp-subset-add-mset*)[]
subgoal
by (*metis add-diff-cancel-left′ distinct-mset-in-diff hp-child-None-notin option.distinct*(*1*) *union-iff*)
subgoal
by (*metis add-diff-cancel-left′ distinct-mset-in-diff hp-child-None-notin option.distinct*(*1*) *union-iff*)
subgoal
by (*metis add-diff-cancel-left′ distinct-mset-in-diff hp-child-None-notin option.distinct*(*1*) *union-iff*)

by (*auto simp add*: *hp-next-children.simps*(*1*) *hp-prev-children.simps*(*1*) *distinct-mset-add disjunct-not-in ac-simps*
split: *if-splits option.splits intro*: *distinct-mset-mono′ union-mset-add-mset-right*
intro: *mset-le-incr-right mset-le-incr-right2 subset-mset-imp-subset-add-mset*)

lemma *distinct-sum-next-prev-child*:
‹*distinct-mset* (*mset-nodes h*) ⟹
distinct-mset ((*if hp-next n h* = *None then* {#} *else* (*mset-nodes* (*the* (*hp-next n h*)))) +
(*if hp-prev n h* = *None then* {#} *else* (*mset-nodes* (*the* (*hp-prev n h*)))) +
(*if hp-child n h* = *None then* {#} *else* (*mset-nodes* (*the* (*hp-child n h*)))))›
using *sum-next-prev-child-subset*[*of h n*] **by** (*auto intro*: *distinct-mset-mono*)

lemma *node-remove-key-in-mset-nodes*:
‹*remove-key a h* ≠ *None* ⟹ *mset-nodes* (*the* (*remove-key a h*)) ⊆# *mset-nodes h*›
apply (*induction a h rule*: *remove-key.induct*)
apply (*auto intro*:)
by (*metis node-remove-key-children-in-mset-nodes sum-image-mset-sum-map*)

lemma *no-relative-ancestor-or-notin*: ‹*hp-parent* (*m′*) *h* = *None* ⟹ *hp-prev m′ h* = *None* ⟹
hp-next m′ h = *None* ⟹ *m′* = *node h* ∨ *m′* ∉# *mset-nodes h*›
apply (*induction m′ h rule*: *hp-next.induct*)
apply (*auto simp*: *hp-prev-children-cons-if*
split:*option.splits*)
apply (*metis hp.exhaust-sel hp-next-children-simps*(*1*) *hp-next-children-simps*(*2*) *hp-parent-children-cons hp-parent-simps*(*2*) *hp-prev-simps option.collapse option.simps*(*5*) *option-last-Some-iff*(*2*))
apply (*metis hp-next-children-simps*(*1*) *hp-next-children-simps*(*2*) *hp-next-children-simps*(*3*) *hp-parent-children-cons*

hp-parent-simps(2) hp-prev-cadr-node hp-prev-children-cons-if option.collapse option.simps(2) option.simps(4) option.simps(5))
apply (*metis hp-next-children-simps(1) hp-next-children-simps(2) hp-next-children-simps(3) hp-parent-children-cons hp-parent-simps(2) hp-prev-cadr-node hp-prev-children-cons-if option.case(1) option.collapse option.simps(2) option.simps(5)*)
apply (*metis option.simps(2)*)
apply (*metis option.simps(2)*)
apply (*metis option.simps(2)*)
by (*metis hp.exhaust-sel hp-parent-None-iff-children-None hp-parent-children-cons hp-prev-simps list.sel(1) list.simps(3) option.case-eq-if option.simps(2)*)

lemma *hp-node-in-find-key-children*:
distinct-mset (*sum-list* (*map mset-nodes h*)) $\Longrightarrow$ *find-key-children x h* = *Some m'* $\Longrightarrow$ *a* $\in\#$ *mset-nodes m'* $\Longrightarrow$
hp-node a m' = *hp-node-children a h*
apply (*induction x h arbitrary*: *m' rule*: *find-key-children.induct*)
apply (*auto split*: *if-splits option.splits simp*: *hp-node-children-Cons-if disjunct-not-in distinct-mset-add*)
by (*metis hp-node-children-simps2*)

lemma *hp-node-in-find-key0*:
distinct-mset (*mset-nodes h*) $\Longrightarrow$ *find-key x h* = *Some m'* $\Longrightarrow$ *a* $\in\#$ *mset-nodes m'* $\Longrightarrow$
hp-node a m' = *hp-node a h*
using *hp-node-in-find-key-children*[*of* ‹*hps h*› *x m' a*]
apply (*cases h*)
apply (*auto split*: *if-splits simp*: *hp-node-children-Cons-if*)
by (*metis hp-node-None-notin2 hp-node-children-None-notin hp-node-children-simps2 sum-image-mset-sum-map*)

lemma *hp-node-in-find-key*:
distinct-mset (*mset-nodes h*) $\Longrightarrow$ *find-key x h* $\neq$ *None* $\Longrightarrow$ *a* $\in\#$ *mset-nodes* (*the* (*find-key x h*)) $\Longrightarrow$
hp-node a (*the* (*find-key x h*)) = *hp-node a h*
using *hp-node-in-find-key0*[*of h x - a*]
by *auto*

context *hmstruct-with-prio*
begin

definition *hmrel* :: ‹(('*a multiset* × ('*a*, '*v*) *hp option*) × ('*a multiset* × '*a multiset* × ('*a* ⇒ '*v*))) *set*›
where
‹*hmrel* = {((ℬ, *xs*), (𝒜, *b*, *w*)). *invar xs* ∧ *distinct-mset b* ∧ 𝒜 = ℬ ∧
((*xs* = *None* ∧ *b* = {#}) ∨
(*xs* ≠ *None* ∧ *b* = *mset-nodes* (*the xs*) ∧
(∀ *v*∈#*b*. *hp-node v* (*the xs*) ≠ *None*) ∧
(∀ *v*∈#*b*. *score* (*the* (*hp-node v* (*the xs*))) = *w v*)))}›

lemma *hp-score-children-iff-hp-score*: ‹*xa* ∈# *sum-list* (*map mset-nodes list*) ⟹ *distinct-mset* (*sum-list* (*map mset-nodes list*)) ⟹
hp-score-children xa list ≠ *None* ⟷ (∃ *x*∈*set list*. *hp-score xa x* ≠ *None* ∧ *hp-score-children xa list* = *hp-score xa x* ∧ (∀ *x*∈*set list* − {*x*}. *hp-score xa x* = *None*))›
apply (*induction list*)
apply (*auto simp*: *hp-node-children-Cons-if*)
apply (*metis disjunct-not-in distinct-mset-add mset-subset-eqD mset-subset-eq-add-left sum-list-map-remove1*)
apply (*metis disjunct-not-in distinct-mset-add mset-subset-eqD mset-subset-eq-add-left sum-list-map-remove1*)
using *WB-List-More.distinct-mset-union2* **apply** *blast*
done

lemma *hp-score-children-in-iff*: ‹*xa* ∈# *sum-list* (*map mset-nodes list*) ⟹ *distinct-mset* (*sum-list* (*map mset-nodes list*)) ⟹
the (*hp-score-children xa list*) ∈ *A* ⟷ (∃ *x*∈*set list*. *hp-score xa x* ≠ *None* ∧ *the* (*hp-score xa x*) ∈ *A*)›
using *hp-score-children-iff-hp-score*[*of xa list*]
by (*auto simp*: *hp-node-children-Cons-if hp-node-children-None-notin2*)

lemma *set-hp-is-hp-score-mset-nodes*:
assumes ‹*distinct-mset* (*mset-nodes a*)›
shows ‹*set-hp a* = (λ*v'*. *the* (*hp-score v' a*)) ‘ *set-mset* (*mset-nodes a*)›
using *assms*
proof (*induction a rule*: *mset-nodes.induct*)
case (*1 x1a x2a x3a*) **note** *p* = *this*(*1*) **and** *dist* = *this*(*2*)
show *?case* (**is** *?A* = *?B*)
proof (*standard*; *standard*)
fix *x*
assume *xA*: ‹*x* ∈ *?A*›
moreover have ‹⋃ (*set-hp* ‘ *set x3a*) = (λ*v'*. *the* (*hp-score-children v' x3a*)) ‘ *set-mset* (∑# (*mset-nodes* ‘# *mset x3a*))› (**is** ‹*?X* = *?Y*›)
proof −
have [*simp*]: ‹*x* ∈ *set x3a* ⟹ *distinct-mset* (*mset-nodes x*)› **for** *x*
using *dist* **by** (*simp add*: *distinct-mset-add sum-list-map-remove1*)
have ‹*?X* = (⋃*x*∈*set x3a*. (λ*v'*. *the* (*hp-score v' x*)) ‘ *set-mset* (*mset-nodes x*))›
using *p dist* **by** (*simp add*: *distinct-mset-add sum-list-map-remove1*)
also have ‹... = (⋃*x*∈*set x3a*. (λ*v'*. *the* (*hp-score-children v' x3a*)) ‘ *set-mset* (*mset-nodes x*))›
apply (*rule SUP-cong*)
apply *simp*
apply (*auto intro*!: *imageI dest*!: *split-list-first simp*: *image-Un cong*: *image-cong*)
apply (*subst image-cong*)
apply (*rule refl*)
apply (*subst hp-node-children-append*(*1*))
using *dist* **apply** *auto*[]
apply (*metis WB-List-More.distinct-mset-union2 add-diff-cancel-right' distinct-mset-in-diff hp-node-children-None-notin2 sum-image-mset-sum-map*)
apply (*rule refl*)
apply *auto*[]
apply (*subst hp-node-children-append*(*1*))
using *dist* **apply** *auto*[]
apply (*metis WB-List-More.distinct-mset-union2 add-diff-cancel-right' distinct-mset-in-diff hp-node-children-None-notin2 sum-image-mset-sum-map*)
apply *auto*
done
also have ‹... = *?Y*›
apply (*auto simp add*: *sum-list-map-remove1*)
by (*metis* (*no-types*, *lifting*) *dist distinct-mset-add hp-node-None-notin2 hp-node-children-None-notin2 hp-score-children-iff-hp-score image-iff mset-nodes.simps option.map-disc-iff sum-image-mset-sum-map*)
finally show *?thesis* **.**
qed
ultimately have ‹*x*=*x2a* ∨ *x* ∈ *?Y*›
by *simp*
then show ‹*x* ∈ *?B*›
apply *auto*
by (*metis* (*no-types*, *lifting*) *1*(*2*) *add-mset-disjoint*(*1*) *distinct-mset-add hp-node-children-simps2 mset-nodes-simps rev-image-eqI*)
next

fix *x*
assume *xA*: ‹$x \in$ *?B*›
moreover have ‹$\bigcup$ (*set-hp* ‘ *set x3a*) = ($\lambda v'$. *the* (*hp-score-children v' x3a*)) ‘ *set-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *mset x3a*))› (**is** ‹*?X* = *?Y*›)
proof −
have [*simp*]: ‹$x \in$ *set x3a* $\Longrightarrow$ *distinct-mset* (*mset-nodes x*)› **for** *x*
using *dist* **by** (*simp add*: *distinct-mset-add sum-list-map-remove1*)
have ‹*?X* = ($\bigcup x \in$*set x3a*. ($\lambda v'$. *the* (*hp-score v' x*)) ‘ *set-mset* (*mset-nodes x*))›
using *p dist* **by** (*simp add*: *distinct-mset-add sum-list-map-remove1*)
also have ‹... = ($\bigcup x \in$*set x3a*. ($\lambda v'$. *the* (*hp-score-children v' x3a*)) ‘ *set-mset* (*mset-nodes x*))›
apply (*rule SUP-cong*)
apply *simp*
apply (*auto intro*!: *imageI dest*!: *split-list-first simp*: *image-Un cong*: *image-cong*)
apply (*subst image-cong*)
apply (*rule refl*)
apply (*subst hp-node-children-append*(*1*))
using *dist* **apply** *auto*[]
apply (*metis WB-List-More.distinct-mset-union2 add-diff-cancel-right' distinct-mset-in-diff hp-node-children-None-notin2 sum-image-mset-sum-map*)
apply (*rule refl*)
apply *auto*[]
apply (*subst hp-node-children-append*(*1*))
using *dist* **apply** *auto*[]
apply (*metis WB-List-More.distinct-mset-union2 add-diff-cancel-right' distinct-mset-in-diff hp-node-children-None-notin2 sum-image-mset-sum-map*)
apply *auto*
done
also have ‹... = *?Y*›
apply (*auto simp add*: *sum-list-map-remove1*)
by (*metis* (*no-types*, *lifting*) *dist distinct-mset-add hp-node-None-notin2 hp-node-children-None-notin2 hp-score-children-iff-hp-score image-iff mset-nodes.simps option.map-disc-iff sum-image-mset-sum-map*)
finally show *?thesis* **.**
qed
ultimately have ‹*x*=*x2a* $\vee$ $x \in$ *?X*›
apply *auto*
by (*metis* (*no-types*, *lifting*) *1*(*2*) *add-mset-disjoint*(*1*) *distinct-mset-add hp-node-children-simps2 image-iff mset-nodes.simps sum-image-mset-sum-map*)
then show ‹$x \in$ *?A*›
by *auto*
qed
qed

definition *mop-get-min2* :: ‹-› **where**
‹*mop-get-min2* = (λ($\mathcal{B}$, *x*). *do* {
ASSERT ($x \neq$ *None*);
RETURN (*get-min2 x*)
})›

lemma *get-min2-mop-prio-peek-min*:
‹(*xs*, *ys*) $\in$ *hmrel* $\Longrightarrow$ *fst ys* $\neq$ {#} $\Longrightarrow$
mop-get-min2 xs $\leq$ $\Downarrow$(*Id*) (*mop-prio-peek-min ys*)›
unfolding *mop-prio-peek-min-def hmrel-def prio-peek-min-def mop-get-min2-def*
apply *refine-vcg*
subgoal
by (*cases xs*; *cases* ‹*the* (*snd xs*)›) *auto*
subgoal

by (*cases xs*; *cases* ‹*the* (*snd xs*)›) *auto*
subgoal
using *set-hp-is-hp-score-mset-nodes*[*of* ‹*hd* (*hps* (*the* (*snd xs*)))›]
apply (*cases xs*; *cases* ‹*the* (*snd xs*)›)
apply (*auto simp*: *invar-def*)
using *le* **apply** *blast*
apply (*cases* ‹*hps* (*the* (*snd xs*))›)
apply *simp*
apply (*auto split*: *if-splits option.splits simp*: *distinct-mset-union in-mset-sum-list-iff dest*!: *split-list*)
apply (*metis* (*no-types*, *lifting*) *hp-node-None-notin2 mem-simps*(*3*) *option.exhaust-sel option.map-sel*)
by (*smt* (*z3*) *diff-union-cancelR distinct-mset-add distinct-mset-in-diff hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-append*(*1*) *hp-node-children-simps*(*3*) *hp-node-children-simps2 mset-map option.map-sel rev-image-eqI set-hp-is-hp-score-mset-nodes set-mset-union sum-mset-sum-list union-iff*)
done

lemma *get-min2-mop-prio-peek-min2*:
‹(*xs*, *ys*) ∈ *hmrel* ⟹
mop-get-min2 xs ≤ ⇓{(*a*,*b*). (*a*,*b*)∈*Id* ∧ *b* = *get-min2* (*snd xs*)} (*mop-prio-peek-min ys*)›
unfolding *mop-prio-peek-min-def hmrel-def prio-peek-min-def mop-get-min2-def*
apply *refine-vcg*
subgoal
by (*cases xs*; *cases* ‹*the* (*snd xs*)›) *auto*
subgoal
using *set-hp-is-hp-score-mset-nodes*[*of* ‹*hd* (*hps* (*the* (*snd xs*)))›]
unfolding *conc-fun-RES*
apply (*cases xs*; *cases* ‹*the* (*snd xs*)›)
apply (*auto simp*: *invar-def*)
using *le* **apply** *blast*
apply (*cases* ‹*hps* (*the* (*snd xs*))›)
apply *simp*
apply (*auto split*: *if-splits option.splits simp*: *distinct-mset-union in-mset-sum-list-iff dest*!: *split-list*)
apply (*metis* (*no-types*, *lifting*) *hp-node-None-notin2 mem-simps*(*3*) *option.exhaust-sel option.map-sel*)
by (*smt* (*z3*) *diff-union-cancelR distinct-mset-add distinct-mset-in-diff hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-append*(*1*) *hp-node-children-simps*(*3*) *hp-node-children-simps2 mset-map option.map-sel rev-image-eqI set-hp-is-hp-score-mset-nodes set-mset-union sum-mset-sum-list union-iff*)
done

lemma *del-min-None-iff*: ‹*del-min a* = *None* ⟷ *a* = *None* ∨ *hps* (*the a*) = []›
by (*cases a*; *cases* ‹*the a*›) (*auto simp*: *pass12-merge-pairs*)

lemma *score-hp-node-pass*$_1$: ‹*distinct-mset* (*sum-list* (*map mset-nodes x3*)) ⟹ *score* (*the* (*hp-node-children v* (*pass*$_1$ *x3*))) = *score* (*the* (*hp-node-children v x3*))›
apply (*induction x3 rule*: *pass*$_1$*.induct*)
subgoal by *auto*
subgoal by *auto*
subgoal for *h1 h2 hs*
apply (*cases h1*; *cases h2*)
apply (*auto simp*: *hp-node-children-Cons-if split*: *option.splits*)
using *WB-List-More.distinct-mset-union2* **apply** *blast*
apply (*metis hp-node-children-None-notin2 sum-image-mset-sum-map*)
by (*metis diff-union-cancelL distinct-mset-in-diff union-iff*)
done

lemma *node-pass*$_2$*-in-nodes*: ‹*pass*$_2$ *hs* ≠ *None* ⟹ *mset-nodes* (*the* (*pass*$_2$ *hs*)) ⊆# *sum-list* (*map mset-nodes hs*)›
 by (*induction hs rule*: *pass*$_2$*.induct*) (*fastforce split*: *option.splits simp del*: *mset-nodes-pass*$_2$)+

lemma *score-pass2-same*:
 ‹*distinct-mset* (*sum-list* (*map mset-nodes x3*)) ⟹ *pass*$_2$ *x3* ≠ *None* ⟹*v* ∈# *sum-list* (*map mset-nodes x3*) ⟹
 score (*the* (*hp-node v* (*the* (*pass*$_2$ *x3*)))) = *score* (*the* (*hp-node-children v x3*))›
 apply (*induction x3 rule*: *pass*$_2$*.induct*)
 subgoal by *auto*
 subgoal for *h hs*
  using *node-pass*$_2$*-in-nodes*[*of hs*]
  apply (*cases h*; *cases* ‹*the* (*pass*$_2$ *hs*)›)
   apply (*auto split*: *option.splits simp*: *hp-node-children-None-notin2 hp-node-children-Cons-if distinct-mset-add*)
  apply (*metis mset-subset-eq-insertD option-last-Nil option-last-Some-iff*(*1*) *pass*$_2$*.simps*(*1*))
  apply (*metis disjunct-not-in mset-subset-eq-insertD not-Some-eq pass*$_2$*.simps*(*1*))
  apply (*metis mset-subset-eq-insertD option-last-Nil option-last-Some-iff*(*1*) *pass*$_2$*.simps*(*1*))
  apply (*metis disjunct-not-in mset-subset-eq-insertD not-Some-eq pass*$_2$*.simps*(*1*))
  apply (*metis disjunct-not-in hp-node-None-notin2 hp-node-children-simps2 mset-nodes-pass*$_2$ *not-Some-eq option.sel*)
  apply (*metis option.distinct*(*2*) *pass*$_2$*.simps*(*1*))
  apply (*metis option.distinct*(*2*) *pass*$_2$*.simps*(*1*))
  apply (*metis option.distinct*(*2*) *pass*$_2$*.simps*(*1*))
  apply (*metis option.distinct*(*2*) *pass*$_2$*.simps*(*1*))
  apply (*meson disjunct-not-in insert-subset-eq-iff*)
  apply (*meson disjunct-not-in insert-subset-eq-iff*)
  apply (*meson disjunct-not-in insert-subset-eq-iff*)
  apply (*meson disjunct-not-in ex-hp-node-children-Some-in-mset-nodes mset-le-add-mset mset-subset-eqD*)
  done
 done

lemma *score-hp-node-merge-pairs-same*: ‹*distinct-mset* (*sum-list* (*map mset-nodes x3*)) ⟹ *v* ∈# *sum-list* (*map mset-nodes x3*) ⟹
 score (*the* (*hp-node v* (*the* (*merge-pairs x3*)))) = *score* (*the* (*hp-node-children v x3*))›
 unfolding *pass12-merge-pairs*[*symmetric*]
 apply (*subst score-pass2-same score-hp-node-pass*$_1$)
 apply *simp-all*
 apply (*metis in-multiset-nempty list.map*(*1*) *mset-nodes-pass*$_1$ *sum-list.Nil*)
 by (*meson score-hp-node-pass*$_1$)
term *mop-get-min2*

definition *mop-hm-pop-min* :: ‹-› **where**
 ‹*mop-hm-pop-min* = (λ(ℬ, *x*). *do* {
  ASSERT (*x* ≠ *None*);
  m ← *mop-get-min2* (ℬ, *x*);
  RETURN (*m*, (ℬ, *del-min x*))
 })›

lemma *get-min2-del-min2-mop-prio-pop-min*:
 assumes ‹(*xs*, *ys*) ∈ *hmrel*›
 shows ‹*mop-hm-pop-min xs* ≤ ⇓(*Id* $\times_r$ *hmrel*) (*mop-prio-pop-min ys*)›
proof −
 have *mop-prio-pop-min-def*: ‹*mop-prio-pop-min ys* = *do* {
  ASSERT (*fst* (*snd ys*)≠{#});

v ← *local.mop-prio-peek-min ys*;
bw ← *mop-prio-del v ys*;
RETURN (*v*, *bw*)
}›
unfolding *mop-prio-pop-min-def local.mop-prio-peek-min-def nres-monad3*
by (*cases ys*) (*auto simp*: *summarize-ASSERT-conv*)
show *?thesis*

using *assms*
unfolding *mop-prio-pop-min-def mop-prio-del-def prio-peek-min-def prio-peek-min-def nres-monad3 case-prod-beta mop-hm-pop-min-def*
apply (*refine-vcg get-min2-mop-prio-peek-min2*)
subgoal by (*auto simp*: *hmrel-def*)
subgoal by *auto*
subgoal
apply (*cases* ‹*the* (*snd xs*)›; *cases xs*)
apply (*auto simp*: *hmrel-def invar-del-min del-min-None-iff pass12-merge-pairs prio-del-def mset-nodes-merge-pairs invar-Some intro*!: *invar-merge-pairs*)
apply (*metis hp-node-children-simps2 score-hp-node-merge-pairs-same*)
apply (*metis list.map*(*1*) *mset-nodes-merge-pairs pairing-heap-assms.merge-pairs-None-iff sum-list.Nil*)
apply (*metis list.map*(*1*) *mset-nodes-merge-pairs pairing-heap-assms.merge-pairs-None-iff sum-list.Nil*)
by (*metis hp-node-children-simps2 score-hp-node-merge-pairs-same*)
done
qed

definition *mop-hm-insert* :: ‹-› **where**
‹*mop-hm-insert* = (λ*w v* (ℬ, *xs*). *do* {
ASSERT (*w* ∈# ℬ ∧ (*xs* ≠ *None* ⟶ *w* ∉# *mset-nodes* (*the xs*)));
RETURN (ℬ, *insert w v xs*)
})›

lemma *mop-prio-insert*:
‹(*xs*, *ys*) ∈ *hmrel* ⟹
mop-hm-insert w v xs ≤ ⇓(*hmrel*) (*mop-prio-insert w v ys*)›
unfolding *mop-prio-insert-def mop-hm-insert-def*
apply *refine-vcg*
subgoal by (*auto simp*: *hmrel-def*)
subgoal by (*auto simp*: *hmrel-def*)
subgoal for *a b*
apply (*auto simp*: *hmrel-def invar-Some php-link le*)
apply (*smt* (*verit*, *del-insts*) *hp.exhaust-sel hp.inject hp-node-children-simps*(*3*) *hp-node-children-simps2 hp-node-simps link.simps node-in-mset-nodes option.sel option-last-Nil option-last-Some-iff*(*2*))
by (*smt* (*verit*, *ccfv-threshold*) *add.right-neutral diff-single-trivial distinct-mset-in-diff hp.sel*(*1*,*2*) *hp-node-None-notin2 hp-node-children-simps*(*2*) *hp-node-children-simps*(*3*) *hp-node-children-simps2 hp-node-node-itself list.simps*(*8*) *mset-nodes-simps option.sel link.simps php.elims*(*2*) *sum-list-simps*(*1*))
done

lemma *find-key-node-itself*[*simp*]: ‹*find-key* (*node y*) *y* = *Some y*›
by (*cases y*) *auto*

lemma *invar-decrease-key*: ‹*le v x* ⟹
invar (*Some* (*Hp w x x3*)) ⟹ *invar* (*Some* (*Hp w v x3*))›
by (*auto simp*: *invar-def intro*!: *transpD*[*OF trans*, *of v x*])

lemma *find-key-children-single*[*simp*]: ‹*find-key-children k* [*x*] = *find-key k x*› by (*cases x*; *auto split*: *option.splits*)

lemma *hp-node-find-key-children*:
‹*distinct-mset* (*sum-list* (*map mset-nodes a*)) ⟹ *find-key-children x a* ≠ *None* ⟹
hp-node x (*the* (*find-key-children x a*)) ≠ *None* ⟹
hp-node x (*the* (*find-key-children x a*)) = *hp-node-children x a*›
apply (*induction x a rule*: *find-key-children.induct*)
apply (*auto split*: *option.splits*)
apply (*metis WB-List-More.distinct-mset-union2 find-key-noneD sum-image-mset-sum-map*)
by (*metis distinct-mset-add find-key-noneD hp-node-None-notin2 hp-node-children-Cons-if hp-node-children-simps2 sum-image-mset-sum-map*)

lemma *hp-node-find-key*:
‹*distinct-mset* (*mset-nodes a*) ⟹ *find-key x a* ≠ *None* ⟹ *hp-score x* (*the* (*find-key x a*)) ≠ *None* ⟹
hp-score x (*the* (*find-key x a*)) = *hp-score x a*›
using *hp-node-find-key-children*[*of* ‹*hps a*› *x*]
apply (*cases a*)
apply *auto*
by (*metis find-key-noneD sum-image-mset-sum-map*)

lemma *score-hp-node-link*:
‹*distinct-mset* (*mset-nodes a* + *mset-nodes b*) ⟹
map-option score (*hp-node w* (*link a b*)) = (*case hp-node w a of Some a* ⇒ *Some* (*score a*)
| - ⇒ *map-option score* (*hp-node w b*))›
apply (*cases a*; *cases b*)
apply (*auto split*: *option.splits*)
by (*metis* (*no-types*, *opaque-lifting*) *distinct-mset-iff ex-hp-node-children-Some-in-mset-nodes mset-add union-mset-add-mset-left union-mset-add-mset-right*)

lemma *hp-node-link-none-iff-parents*: ‹*hp-node va* (*link a b*) = *None* ⟷ *hp-node va a* = *None* ∧ *hp-node va b* = *None*›
by *auto*

lemma *score-hp-node-link2*:
‹*distinct-mset* (*mset-nodes a* + *mset-nodes b*) ⟹ (*hp-node w* (*link a b*)) ≠ *None* ⟹
score (*the* (*hp-node w* (*link a b*))) = (*case hp-node w a of Some a* ⇒ (*score a*)
| - ⇒ *score* (*the* (*hp-node w b*)))›
using *score-hp-node-link*[*of a b w*] **by** (*cases* ‹*hp-node w* (*link a b*)›; *cases* ‹*hp-node w b*›)
(*auto split*: *option.splits*)

definition *mop-hm-decrease-key* :: ‹-› **where**
‹*mop-hm-decrease-key* = (λ*w v* (𝓑, *xs*). *do* {
ASSERT (*w* ∈# 𝓑);
if xs = *None then RETURN* (𝓑, *xs*)
else RETURN (𝓑, *decrease-key w v* (*the xs*))
})›

lemma *decrease-key-mop-prio-change-weight*:
assumes ‹(*xs*, *ys*) ∈ *hmrel*›
shows ‹*mop-hm-decrease-key w v xs* ≤ ⇓(*hmrel*) (*mop-prio-change-weight w v ys*)›
proof −
let *?w* = ‹*snd* (*snd ys*)›
let *?xs* = ‹*snd xs*›
have *K*: ‹*xs'* = *?xs* ⟹ *node* (*the xs'*) ≠ *w* ⟷ *remove-key w* (*the ?xs*)≠ *None*› **for** *xs'*
using *assms* **by** (*cases xs*; *cases* ‹*the ?xs*›) (*auto simp*: *hmrel-def*)

have [*simp*]: ‹*add-mset* (*node x2a*) (*sum-list* (*map mset-nodes* (*hps x2a*))) = *mset-nodes x2a*› **for** *x2a*
  by (*cases x2a*) *auto*
have *f*: ‹*w* ∈# *fst* (*snd ys*) ⟹ *find-key w* (*the ?xs*) = *Some* (*Hp w* (*?w w*) (*hps* (*the* (*find-key w* (*the ?xs*)))))›
  using *assms invar-find-key*[*of* ‹*the ?xs*› *w*] *find-key-None-or-itself*[*of w* ‹*the ?xs*›]
    find-key-none-iff[*of w* ‹[*the ?xs*]›]
    hp-node-find-key[*of* ‹*the ?xs*› *w*]
  apply (*cases* ‹*the* (*find-key w* (*the ?xs*))›; *cases* ‹*find-key w* (*the ?xs*)›)
  apply *simp-all*
  apply (*auto simp*: *hmrel-def invar-Some*)
  by (*metis hp-node-None-notin2 option.map-sel option.sel*)

then have ‹*w* ∈# *fst* (*snd ys*) ⟹ *invar* (*Some* (*Hp w* (*?w w*) (*hps* (*the* (*find-key w* (*the ?xs*))))))›
  using *assms invar-find-key*[*of* ‹*the ?xs*› *w*] **by** (*auto simp*: *hmrel-def invar-Some*)
moreover have ‹*w* ∈# *fst* (*snd ys*) ⟹ *find-key w* (*the ?xs*) ≠ *None* ⟹ *remove-key w* (*the ?xs*) ≠ *None* ⟹
  distinct-mset (*mset-nodes* (*Hp w v* (*hps* (*the* (*find-key w* (*the ?xs*))))) + *mset-nodes* (*the* (*remove-key w* (*the ?xs*))))›
  using *assms distinct-mset-find-node-next*[*of* ‹*the ?xs*› *w* ‹*the* (*find-key w* (*the ?xs*))›]
  apply (*subst* ‹*w* ∈# *fst* (*snd ys*) ⟹ *find-key w* (*the ?xs*) = *Some* (*Hp w* (*?w w*) (*hps* (*the* (*find-key w* (*the ?xs*)))))›) **apply** (*auto simp*: *hmrel-def*)
  apply (*metis* ‹⋀*x2a*. *add-mset* (*node x2a*) (*sum-list* (*map mset-nodes* (*hps x2a*))) = *mset-nodes x2a*› *distinct-mset-add distinct-mset-add-mset find-key-None-or-itself option.distinct*(*1*) *option.sel*)
  apply (*metis find-key-None-or-itself in-find-key-notin-remove-key node-in-mset-nodes option.distinct*(*1*) *option.sel*)
  by (*metis* (*no-types*, *opaque-lifting*) *Groups.add-ac*(*2*) ‹⋀*x2a*. *add-mset* (*node x2a*) (*sum-list* (*map mset-nodes* (*hps x2a*))) = *mset-nodes x2a*› *distinct-mset-add-mset find-remove-mset-nodes-full union-mset-add-mset-right*)
ultimately show *?thesis*
  using *assms*
  unfolding *mop-prio-change-weight-def mop-hm-decrease-key-def*
  apply *refine-vcg*
  subgoal by (*auto simp*: *hmrel-def*)
  subgoal by (*auto simp*: *hmrel-def*)
  subgoal
    using *invar-decrease-key*[*of v* ‹*?w w*› *w* ‹*hps* (*the* (*find-key w* (*the ?xs*)))›]
      find-key-none-iff[*of w* ‹[*the ?xs*]›] *find-key-None-or-itself*[*of w* ‹*the ?xs*›]
      invar-find-key[*of* ‹*the ?xs*› *w*] *hp-node-find-key*[*of* ‹*the ?xs*› *w*] *f*
      find-remove-mset-nodes-full[*of* ‹*the ?xs*› *w* ‹*the* (*remove-key w* (*the ?xs*))› ‹*the*(*find-key w* (*the ?xs*))›]
      hp-node-in-find-key0[*of* ‹*the ?xs*› *w* ‹*the*(*find-key w* (*the ?xs*))›]
    apply (*auto simp*: *hmrel-def decrease-key-def remove-key-None-iff invar-def score-hp-node-link2*
      simp del: *find-key-none-iff php.simps*
      intro:
      split: *option.splits hp.splits*)
    apply (*metis hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-simps2 sum-image-mset-sum-map*)
    apply (*metis hp-node-children-simps2*)
    apply (*metis invar-Some php-link php-remove-key*)
    apply (*metis union-ac*(*2*))
    apply (*metis member-add-mset union-iff*)

    using *K* **apply** (*solves* ‹*simp add*: *score-hp-node-link2 del*: *php.simps*›)

    using *K* **apply** (*simp add*: *score-hp-node-link2 del*: *php.simps*)
    apply (*subst score-hp-node-link2*)
    apply (*solves simp*)
    apply (*simp add*: *hp-node-link-none-iff-parents*)

apply (*auto split*: *option.splits*)
apply (*metis member-add-mset mset-cancel-union*(*2*))
apply (*smt* (*verit*, *ccfv-threshold*) *hp-node-None-notin2 hp-node-children-None-notin2 hp-score-remove-key-other map-option-is-None member-add-mset option.map-sel option.sel option-hd-Nil option-hd-Some-iff*(*2*) *sum-image-mset-sum-map union-iff*)
by (*metis hp-node-None-notin2 hp-node-children-simps2 hp-node-in-find-key0 option.sel option-hd-Nil option-hd-Some-iff*(*2*))
done
qed

lemma *pass*$_1$*-empty-iff*[*simp*]: ‹*pass*$_1$ *x* = [] ⟷ *x*= []›
by (*cases x rule*: *pass*$_1$*.cases*) *auto*

lemma *sum-list-map-mset-nodes-empty-iff*[*simp*]: ‹*sum-list* (*map mset-nodes x3*) = {#} ⟷ *x3* = []›
by (*cases x3*; *cases* ‹*hd x3*›) *auto*

lemma *hp-score-link*:
‹*a* ∈# *mset-nodes h1* ⟹ *distinct-mset* (*mset-nodes h1* + *mset-nodes h2*) ⟹ *hp-score a* (*link h1 h2*) = *hp-score a h1*›
apply (*cases h1*; *cases h2*)
apply (*auto split*: *option.splits simp add*: *hp-node-children-None-notin2*)
by (*metis diff-union-cancelL distinct-mem-diff-mset ex-hp-node-children-Some-in-mset-nodes hp-node-children-simps2*)

lemma *hp-score-link-skip-first*[*simp*]:
‹*a* ∉# *mset-nodes h1* ⟹ *hp-score a* (*link h1 h2*) = *hp-score a h2*›
by (*cases h1*; *cases h2*)
(*auto split*: *option.splits simp add*: *hp-node-children-None-notin2*)

lemma *hp-score-merge-pairs*:
‹*distinct-mset* (*sum-list* (*map mset-nodes ys*)) ⟹ *merge-pairs ys* ≠ *None* ⟹
hp-score a (*the* (*merge-pairs* (*ys*))) = *hp-score-children a* (*ys*)›
apply (*induction ys rule*: *pass*$_1$*.induct*)
apply (*auto simp add*: *hp-node-children-Cons-if Let-def split*: *option.splits*)
apply (*simp add*: *disjunct-not-in distinct-mset-add hp-score-link*)
apply (*subst hp-score-link*)
apply *simp*
apply *simp*
apply (*metis mset-nodes-merge-pairs option.sel option-hd-Nil option-hd-Some-iff*(*2*) *union-assoc*)
apply (*metis Groups.add-ac*(*1*) *distinct-mset-union hp-score-link*)
apply (*subst hp-score-link*)
apply *simp*
apply *simp*
apply (*metis mset-nodes-merge-pairs option.sel option-hd-Nil option-hd-Some-iff*(*2*) *union-assoc*)
apply (*meson hp-score-link-skip-first*)
apply (*subst hp-score-link*)
apply *simp*
apply *simp*
apply (*metis mset-nodes-merge-pairs option.sel option-hd-Nil option-hd-Some-iff*(*2*) *union-assoc*)
apply (*metis Groups.add-ac*(*1*) *distinct-mset-union hp-score-link*)
by (*metis Duplicate-Free-Multiset.distinct-mset-union2 merge-pairs-None-iff option.simps*(*2*))

definition *decrease-key2* **where**
‹*decrease-key2 a w h* = (*if h* = *None then None else decrease-key a w* (*the h*))›
lemma *hp-mset-rel-def*: ‹*hmrel* = {((ℬ, *h*), (𝒜, *m*, *w*)). *distinct-mset m* ∧ 𝒜=ℬ ∧

(*h* = *None* ⟷ *m* = {#}) ∧
(*m* ≠ {#} ⟶ (*mset-nodes* (*the h*) = *m* ∧ (∀ *a*∈#*m*. *Some* (*w a*) = *hp-score a* (*the h*)) ∧ *invar h*))}›
unfolding *hmrel-def*
apply (*auto simp*:)
apply (*metis in-multiset-nempty node-in-mset-nodes*)
apply (*simp add*: *option.expand option.map-sel*)
apply (*metis Some-to-the hp-node-None-notin2 option.map-sel*)
by (*metis Some-to-the hp-node-None-notin2 option.map-sel*)

lemma (**in** −)*find-key-None-remove-key-ident*: ‹*find-key a h* = *None* ⟹ *remove-key a h* = *Some h*›
by (*induction a h rule*: *find-key.induct*)
(*auto split*: *if-splits*)

lemma *decrease-key2*:
assumes ‹(*x*, *m*) ∈ *hmrel*› ‹(*a*,*a'*)∈*Id*› ‹(*w*,*w'*)∈*Id*› ‹*le w* (*snd* (*snd m*) *a*)›
shows ‹*mop-hm-decrease-key a w x* ≤ ⇓ (*hmrel*) (*mop-prio-change-weight a' w' m*)›
proof −
show *?thesis*
using *assms*
unfolding *decrease-key2-def*
mop-prio-insert-def mop-prio-change-weight-def mop-hm-decrease-key-def
apply *refine-rcg*
subgoal by (*auto simp*: *hmrel-def*)
subgoal
using *php-decrease-key*[*of* ‹*the* (*snd x*)› *w a*]
apply (*auto simp*: *hp-mset-rel-def decrease-key-def invar-def split*: *option.splits hp.splits*)
apply (*metis find-key-None-remove-key-ident in-remove-key-changed option.sel option.simps*(*2*))
apply (*metis empty-neutral*(*1*) *find-key-head-node-iff hp.sel*(*1*) *mset-map mset-nodes.simps option.simps*(*1*) *remove-key-None-iff sum-mset-sum-list union-mset-add-mset-left*)
apply (*metis find-key-node-itself hp.sel*(*1*) *hp-node-children-simps2 option.sel remove-key-None-iff*)
apply (*metis find-key-node-itself find-key-notin hp.sel*(*2*) *hp-node-node-itself option.distinct*(*1*) *option.map-sel option.sel remove-key-None-iff*)
apply (*metis invar-Some invar-find-key php.simps*)
apply (*metis* (*no-types*, *lifting*) *add-mset-add-single find-key-None-or-itself find-remove-mset-nodes-full hp.sel*(*1*) *mset-nodes-simps option.sel option.simps*(*2*) *union-commute union-mset-add-mset-left*)
apply (*smt* (*verit*) *Duplicate-Free-Multiset.distinct-mset-mono disjunct-not-in distinct-mset-add find-key-None-or-itself hp.sel*(*1*) *hp.sel*(*2*) *hp-node-simps hp-score-link in-find-key-notin-remove-key mset-nodes.simps mset-nodes-find-key-subset node-remove-key-in-mset-nodes option.distinct*(*1*) *option.sel option.simps*(*9*) *union-iff union-single-eq-member*)
apply (*smt* (*verit*) *disjunct-not-in distinct-mset-add find-key-None-or-itself find-remove-mset-nodes-full hp.sel*(*1*) *hp-node-None-notin2 hp-node-children-simps2 hp-node-in-find-key0 hp-score-link hp-score-link-skip-first hp-score-remove-key-other map-option-is-None mset-nodes-simps option.distinct*(*1*) *option.sel union-iff*)
by (*metis find-key-None-or-itself find-key-notin hp.sel*(*1*) *hp.sel*(*2*) *hp-node-find-key hp-node-simps option.distinct*(*1*) *option.sel option.simps*(*9*))
done
qed

end

interpretation *ACIDS*: *hmstruct-with-prio* **where**
le = ‹(≥) :: *nat* ⇒ *nat* ⇒ *bool*› **and**
lt = ‹(>)›
apply *unfold-locales*
subgoal by *auto*
subgoal by *auto*

subgoal by (*auto simp*: *transp-def*)
subgoal by (*auto simp*: *totalp-on-def*)
done

end
theory *Relational-Pairing-Heaps*
imports *Pairing-Heaps*
begin

1.1.2 Flat Version of Pairing Heaps

Splitting genealogy to Relations

In this subsection, we replace the tree version by several arrays that represent the relations (parent, child, next, previous) of the same trees.

type-synonym ('*a*, '*b*) *hp-fun* = ‹(('*a* ⇒ '*a option*) × ('*a* ⇒ '*a option*) × ('*a* ⇒ '*a option*) × ('*a* ⇒ '*a option*) × ('*a* ⇒ '*b option*))›

definition *hp-set-all* :: ‹'*a* ⇒ '*a option* ⇒ '*a option* ⇒ '*a option* ⇒ '*a option* ⇒ '*b option* ⇒ ('*a*, '*b*) *hp-fun* ⇒ ('*a*, '*b*) *hp-fun* › **where**
‹*hp-set-all i prev nxt child par sc* = (λ(*prevs*, *nxts*, *childs*, *parents*, *scores*). (*prevs*(*i*:=*prev*), *nxts*(*i*:=*nxt*), *childs*(*i*:=*child*), *parents*(*i*:=*par*), *scores*(*i*:=*sc*)))›

definition *hp-update-prev* :: ‹'*a* ⇒ '*a option* ⇒ ('*a*, '*b*) *hp-fun* ⇒ ('*a*, '*b*) *hp-fun*› **where**
‹*hp-update-prev i prev* = (λ(*prevs*, *nxts*, *childs*, *parents*, *score*). (*prevs*(*i*:=*prev*), *nxts*, *childs*, *parents*, *score*))›

definition *hp-update-nxt* :: ‹'*a* ⇒ '*a option* ⇒ ('*a*, '*b*) *hp-fun* ⇒ ('*a*, '*b*) *hp-fun*› **where**
‹*hp-update-nxt i nxt* = (λ(*prevs*, *nxts*, *childs*, *parents*, *score*). (*prevs*, *nxts*(*i*:=*nxt*), *childs*, *parents*, *score*))›

definition *hp-update-parents* :: ‹'*a* ⇒ '*a option* ⇒ ('*a*, '*b*) *hp-fun* ⇒ ('*a*, '*b*) *hp-fun*› **where**
‹*hp-update-parents i nxt* = (λ(*prevs*, *nxts*, *childs*, *parents*, *score*). (*prevs*, *nxts*, *childs*, *parents*(*i*:=*nxt*), *score*))›

definition *hp-update-score* :: ‹'*a* ⇒ '*b option* ⇒ ('*a*, '*b*) *hp-fun* ⇒ ('*a*, '*b*) *hp-fun*› **where**
‹*hp-update-score i nxt* = (λ(*prevs*, *nxts*, *childs*, *parents*, *score*). (*prevs*, *nxts*, *childs*, *parents*, *score*(*i*:=*nxt*)))›

fun *hp-read-nxt* :: ‹- ⇒ ('*a*, '*b*) *hp-fun* ⇒ -› **where** ‹*hp-read-nxt i* (*prevs*, *nxts*, *childs*) = *nxts i*›
fun *hp-read-prev* :: ‹- ⇒ ('*a*, '*b*) *hp-fun* ⇒ -› **where** ‹*hp-read-prev i* (*prevs*, *nxts*, *childs*) = *prevs i*›
fun *hp-read-child* :: ‹- ⇒ ('*a*, '*b*) *hp-fun* ⇒ -› **where** ‹*hp-read-child i* (*prevs*, *nxts*, *childs*, *parents*, *scores*) = *childs i*›
fun *hp-read-parent* :: ‹- ⇒ ('*a*, '*b*) *hp-fun* ⇒ -› **where** ‹*hp-read-parent i* (*prevs*, *nxts*, *childs*, *parents*, *scores*) = *parents i*›
fun *hp-read-score* :: ‹- ⇒ ('*a*, '*b*) *hp-fun* ⇒ -› **where** ‹*hp-read-score i* (*prevs*, *nxts*, *childs*, *parents*, *scores*) = *scores i*›

It was not entirely clear from the ground up whether we would actually need to have the conditions of emptyness of the previous or the parent. However, these are the only conditions to know whether a node is in the treen or not, so we decided to include them. It is critical to not add that the scores are empty, because this is the only way to track the scores after removing a node.

We initially inlined the definition of *empty-outside*, but the simplifier immediatly hung himself.

definition *empty-outside* :: ‹-› **where**

‹*empty-outside* $\mathcal{V}$ *prevs* = (∀ *x*. *x* ∉# $\mathcal{V}$ ⟶ *prevs x* = *None*)›

definition *encoded-hp-prop* :: ‹′*e multiset* ⇒ (′*e*,′*f*) *hp multiset* ⇒ (′*e*, ′*f*) *hp-fun* ⇒ -› **where**
‹*encoded-hp-prop* $\mathcal{V}$ *m* = (λ(*prevs*,*nxts*,*children*, *parents*, *scores*). *distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# *m*)) ∧
set-mset ($\sum_{\#}$ (*mset-nodes* '# *m*)) ⊆ *set-mset* $\mathcal{V}$ ∧
(∀ *m*∈# *m*. ∀ *x* ∈# *mset-nodes m*. *prevs x* = *map-option node* (*hp-prev x m*)) ∧
(∀ *m*′∈#*m*. ∀ *x* ∈# *mset-nodes m*′. *nxts x* = *map-option node* (*hp-next x m*′)) ∧
(∀ *m*∈# *m*. ∀ *x* ∈# *mset-nodes m*. *children x* = *map-option node* (*hp-child x m*)) ∧
(∀ *m*∈# *m*. ∀ *x* ∈# *mset-nodes m*. *parents x* = *map-option node* (*hp-parent x m*)) ∧
(∀ *m*∈# *m*. ∀ *x* ∈# *mset-nodes m*. *scores x* = *map-option score* (*hp-node x m*)) ∧
empty-outside ($\sum_{\#}$ (*mset-nodes* '# *m*)) *prevs* ∧
empty-outside ($\sum_{\#}$ (*mset-nodes* '# *m*)) *parents*)›

lemma *empty-outside-alt-def*: ‹*empty-outside* $\mathcal{V}$ *f* = (*dom f* ∩ *UNIV* − *set-mset* $\mathcal{V}$ = {})›
unfolding *empty-outside-def*
by *auto*

lemma *empty-outside-add-mset*[*simp*]:
‹*f v* = *None* ⟹ *empty-outside* (*add-mset v* $\mathcal{V}$) *f* ⟷ *empty-outside* $\mathcal{V}$ *f*›
unfolding *empty-outside-alt-def*
by *auto*

lemma *empty-outside-notin-None*: ‹*empty-outside* $\mathcal{V}$ *prevs* ⟹ *a* ∉# $\mathcal{V}$ ⟹ *prevs a* = *None*›
unfolding *empty-outside-alt-def*
by *auto*

lemma *empty-outside-update-already-in*[*simp*]: ‹*x* ∈# $\mathcal{V}$ ⟹ *empty-outside* $\mathcal{V}$ (*prevs*(*x* := *a*)) = *empty-outside* $\mathcal{V}$ *prevs*›
unfolding *empty-outside-alt-def*
by *auto*

lemma *encoded-hp-prop-irrelevant*:
assumes ‹*a* ∉# $\sum_{\#}$ (*mset-nodes* '# *m*)› **and** ‹*a*∈#$\mathcal{V}$› **and**
‹*encoded-hp-prop* $\mathcal{V}$ *m* (*prevs*, *nxts*, *children*, *parents*, *scores*)›
shows
‹*encoded-hp-prop* $\mathcal{V}$ (*add-mset* (*Hp a sc* []) *m*) (*prevs*, *nxts*(*a*:=*None*), *children*(*a*:=*None*), *parents*, *scores*(*a*:=*Some sc*))›
using *assms* **by** (*auto simp*: *encoded-hp-prop-def empty-outside-notin-None*)

lemma *hp-parent-single-child*[*simp*]: ‹*hp-parent* (*node a*) (*Hp m* w_m [*a*]) = *Some* (*Hp m* w_m [*a*])›
by (*cases a*) (*auto simp*: *hp-parent.simps*(*1*))

lemma *hp-parent-children-single-hp-parent*[*simp*]: ‹*hp-parent-children b* [*a*] = *hp-parent b a*›
by (*auto simp*: *hp-parent-children-def*)

lemma *hp-parent-single-child-If*:
‹*b* ≠ *m* ⟹ *hp-parent b* (*Hp m* w_m (*a* # [])) = (*if b* = *node a then Some* (*Hp m* w_m [*a*]) *else hp-parent b a*)›
by (*auto simp*: *hp-parent-simps*)

lemma *hp-parent-single-child-If2*:
‹*distinct-mset* (*add-mset m* (*mset-nodes a*)) ⟹
hp-parent b (*Hp m* w_m (*a* # [])) = (*if b* = *m then None else if b* = *node a then Some* (*Hp m* w_m [*a*]) *else hp-parent b a*)›

by (*auto simp*: *hp-parent-simps*)

lemma *hp-parent-single-child-If3*:
‹*distinct-mset* (*add-mset m* (*mset-nodes a* + *sum-list* (*map mset-nodes xs*))) $\Longrightarrow$
hp-parent b (*Hp m* w_m (*a* # *xs*)) = (*if b* = *m then None else if b* = *node a then Some* (*Hp m* w_m (*a* # *xs*)) *else hp-parent-children b* (*a* # *xs*))›
by (*auto simp*: *hp-parent-simps*)

lemma *hp-parent-is-first-child*[*simp*]: ‹*hp-parent* (*node a*) (*Hp m* w_m (*a* # ch_m)) = *Some* (*Hp m* w_m (*a* # ch_m))›
by (*auto simp*: *hp-parent.simps*(*1*))

lemma *hp-parent-children-in-first-child*[*simp*]: ‹*distinct-mset* (*mset-nodes a* + *sum-list* (*map mset-nodes* ch_m)) $\Longrightarrow$
xa ∈# *mset-nodes a* $\Longrightarrow$ *hp-parent-children xa* (*a* # ch_m) = *hp-parent xa a*›
by (*auto simp*: *hp-parent-children-cons split*: *option.splits dest*: *multi-member-split*)

lemma *encoded-hp-prop-link*:
fixes ch_m *a prevs parents m*
defines ‹*prevs'* ≡ (*if* ch_m = [] *then prevs else prevs* (*node* (*hd* ch_m) := *Some* (*node a*)))›
defines ‹*parents'* ≡ (*if* ch_m = [] *then parents else parents* (*node* (*hd* ch_m) := *None*))›
assumes ‹*encoded-hp-prop* $\mathcal{V}$ (*add-mset* (*Hp m* w_m ch_m) (*add-mset a x*)) (*prevs*, *nxts*, *children*, *parents*, *scores*)›
shows
‹*encoded-hp-prop* $\mathcal{V}$ (*add-mset* (*Hp m* w_m (*a* # ch_m)) *x*) (*prevs'*, *nxts*(*node a*:= *if* ch_m = [] *then None else Some* (*node* (*hd* ch_m))),
children(*m* := *Some* (*node a*)), *parents'*(*node a*:= *Some m*), *scores*(*m*:=*Some* w_m))›
proof −
have *H*[*simp*]: ‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + (*mset-nodes a*))› ‹*distinct-mset* (*mset-nodes a*)›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m))› **and**
dist: ‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + (*mset-nodes a*) + $\sum_{\#}$ (*mset-nodes* '# *x*))›
‹*m* ∉# *sum-list* (*map mset-nodes* ch_m) + (*mset-nodes a*) + $\sum_{\#}$ (*mset-nodes* '# *x*)› **and**
incl: ‹*set-mset* ($\sum_{\#}$ (*mset-nodes* '# *add-mset* (*Hp m* w_m ch_m) (*add-mset a x*))) ⊆ *set-mset* $\mathcal{V}$›
using *assms* **unfolding** *encoded-hp-prop-def prod.simps* **apply** *auto*
by (*metis distinct-mset-add mset-nodes-simps sum-mset.insert union-assoc*)+
have [*simp*]: ‹*distinct-mset* (*mset-nodes a* + *sum-list* (*map mset-nodes* ch_m))›
by (*metis H*(*1*) *union-ac*(*2*))
have *1*: ‹ch_m ≠ [] $\Longrightarrow$ *node a* ≠ *node* (*hd* ch_m)›
if ‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + (*mset-nodes a* + $\sum_{\#}$ (*mset-nodes* '# *x*)))›
using *that* **by** (*cases* ch_m; *cases a*; *auto*)
have *K*: ‹*xa* ∈# *mset-nodes a* $\Longrightarrow$ *xa* ∉# *sum-list* (*map mset-nodes* ch_m)›
‹*xa* ∈# *sum-list* (*map mset-nodes* ch_m) $\Longrightarrow$ *xa* ∉# *mset-nodes a*›**for** *xa*
using *H* **by** (*auto simp del*: *H dest*!: *multi-member-split*)

have [*simp*]: ‹ch_m ≠ [] $\Longrightarrow$ *ma* ∈# *x* $\Longrightarrow$ *hp-parent* (*node* (*hd* ch_m)) *ma* = *None*›
‹*ma* ∈# *x* $\Longrightarrow$ *hp-parent* (*node a*) *ma* = *None*›
‹*ma* ∈# *x* $\Longrightarrow$ *node a* ∉# *mset-nodes ma*› **for** *ma*
by (*cases* ch_m; *cases* ‹*hd* ch_m›; *cases a*; *use dist* **in** ‹*auto simp del*: *H dest*!: *multi-member-split*›; *fail*)+
have [*simp*]: ‹*xa* ∈# *sum-list* (*map mset-nodes* ch_m) $\Longrightarrow$ *xa* ≠ *node* (*hd* ch_m) $\Longrightarrow$
(*hp-parent xa* (*Hp m* w_m ch_m)) = (*hp-parent-children xa* (*a* # ch_m))› **for** *xa*
using *dist H*
by (*cases* ch_m; *cases x*)
(*auto simp*: *hp-parent-single-child-If3 hp-parent-children-cons*

simp del: *H*
dest!: *multi-member-split split*: *option.splits*)

show *?thesis*
using *assms 1* **unfolding** *encoded-hp-prop-def prod.simps*
apply (*intro conjI impI ballI*)
subgoal by (*auto simp*: *ac-simps*)
subgoal by (*auto simp*: *ac-simps*)
subgoal
apply (*auto simp*: *prevs′-def hp-prev-skip-hd-children dest*: *multi-member-split*)
by (*metis add-mset-disjoint*(*1*) *distinct-mset-add image-msetI in-Union-mset-iff mset-add node-hd-in-sum union-iff*)
subgoal apply *simp*
apply (*intro conjI impI allI*)
subgoal by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
subgoal by (*auto dest*: *multi-member-split*)[]
subgoal by (*auto dest*!: *multi-member-split*)[]
subgoal
by (*auto dest*: *multi-member-split distinct-mset-union simp*: *hp-next-skip-hd-children*)
done
subgoal
by (*auto split*: *option.splits simp*: *K*)
subgoal
by (*auto simp*: *hp-parent-single-child-If2 hp-parent-single-child-If3*)
subgoal
by (*auto split*: *option.splits simp*: *K*(*2*))
subgoal
by (*auto simp*: *ac-simps*)
subgoal
by (*auto simp*: *ac-simps*)
done
qed

fun *find-first-not-none* **where**
‹*find-first-not-none* (*None* # *xs*) = *find-first-not-none xs*› |
‹*find-first-not-none* (*Some a* # -) = *Some a*›|
‹*find-first-not-none* [] = *None*›

lemma *find-first-not-none-alt-def*:
‹*find-first-not-none xs* = *map-option the* (*option-hd* (*filter* ((≠) *None*) *xs*))›
by (*induction xs rule*: *find-first-not-none.induct*) *auto*

In the following we distinguish between the tree part and the tree part without parent (aka the list part). The latter corresponds to a tree where we have removed the source, but the leafs remains in the correct form. They are different for first level nexts and first level children.

definition *encoded-hp-prop-list* :: ‹′*e multiset* ⇒ (′*e*,′*f*) *hp multiset* ⇒ (′*e*,′*f*) *hp list* ⇒ -› **where**
‹*encoded-hp-prop-list* $\mathcal{V}$ *m xs* = (λ(*prevs*,*nxts*,*children*, *parents*, *scores*). *distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *m* + *mset-nodes* ‘# (*mset xs*))) ∧
set-mset ($\sum_{\#}$ (*mset-nodes* ‘# *m* + *mset-nodes* ‘# (*mset xs*))) ⊆ *set-mset* $\mathcal{V}$ ∧
(∀ *m*′∈#*m*. ∀ *x* ∈# *mset-nodes m*′. *nxts x* = *map-option node* (*hp-next x m*′)) ∧
(∀ *m*∈# *m*. ∀ *x* ∈# *mset-nodes m*. *prevs x* = *map-option node* (*hp-prev x m*)) ∧
(∀ *m*∈# *m*. ∀ *x* ∈# *mset-nodes m*. *children x* = *map-option node* (*hp-child x m*)) ∧
(∀ *m*∈# *m*. ∀ *x* ∈# *mset-nodes m*. *parents x* = *map-option node* (*hp-parent x m*)) ∧
(∀ *m*∈# *m*. ∀ *x* ∈# *mset-nodes m*. *scores x* = *map-option score* (*hp-node x m*)) ∧

($\forall x \in\# \sum_{\#}$ (*mset-nodes* ‘# *mset xs*). *nxts x* = *map-option node* (*hp-next-children x xs*)) ∧
($\forall x \in\# \sum_{\#}$ (*mset-nodes* ‘# *mset xs*). *prevs x* = *map-option node* (*hp-prev-children x xs*)) ∧
($\forall x \in\# \sum_{\#}$ (*mset-nodes* ‘# *mset xs*). *children x* = *map-option node* (*hp-child-children x xs*)) ∧
($\forall x \in\# \sum_{\#}$ (*mset-nodes* ‘# *mset xs*). *parents x* = *map-option node* (*hp-parent-children x xs*)) ∧
($\forall x \in\# \sum_{\#}$ (*mset-nodes* ‘# *mset xs*). *scores x* = *map-option score* (*hp-node-children x xs*)) ∧
empty-outside ($\sum_{\#}$ (*mset-nodes* ‘# *m* + *mset-nodes* ‘# (*mset xs*))) *prevs* ∧
empty-outside ($\sum_{\#}$ (*mset-nodes* ‘# *m* + *mset-nodes* ‘# (*mset xs*))) *parents*)
›

lemma *encoded-hp-prop-list-encoded-hp-prop*[*simp*]: ‹*encoded-hp-prop-list* $\mathcal{V}$ *arr* [] *h* = *encoded-hp-prop* $\mathcal{V}$ *arr h*›
unfolding *encoded-hp-prop-list-def encoded-hp-prop-def* **by** *auto*

lemma *encoded-hp-prop-list-encoded-hp-prop-single*[*simp*]: ‹*encoded-hp-prop-list* $\mathcal{V}$ {#} [*arr*] *h* = *encoded-hp-prop* $\mathcal{V}$ {#*arr*#} *h*›
unfolding *encoded-hp-prop-list-def encoded-hp-prop-def* **by** *auto*

lemma *empty-outside-set-none-outside*[*simp*]: ‹*empty-outside* $\mathcal{V}$ *prevs* ⟹ *a* ∉# $\mathcal{V}$ ⟹ *empty-outside* $\mathcal{V}$ (*prevs*(*a* := *None*))›
unfolding *empty-outside-alt-def* **by** *auto*

lemma *encoded-hp-prop-list-remove-min*:
fixes *parents a child children*
defines ‹*parents'* ≡ (*if children a* = *None then parents else parents*(*the* (*children a*) := *None*))›
assumes ‹*encoded-hp-prop-list* $\mathcal{V}$ (*add-mset* (*Hp a b child*) *xs*) [] (*prevs*, *nxts*, *children*, *parents*, *scores*)›
shows ‹*encoded-hp-prop-list* $\mathcal{V}$ *xs child* (*prevs*(*a*:=*None*), *nxts*, *children*(*a*:=*None*), *parents'*, *scores*)›
proof −
have *a*: ‹*children a* = *None* ⟷ *child* = []› **and**
b: ‹*children a* ≠ *None* ⟹ *the* (*children a*) = *node* (*hd child*)›
using *assms*
unfolding *encoded-hp-prop-list-def*
by (*cases child*; *auto simp*: *ac-simps hp-parent-single-child-If3 dest*: *multi-member-split*; *fail*)+
have *dist*: ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *add-mset* (*Hp a b child*) *xs*))›
using *assms* **unfolding** *encoded-hp-prop-list-def prod.simps*
by (*metis empty-neutral*(*2*) *image-mset-empty mset-zero-iff*)
then have ‹*child* ≠ [] ⟹ *distinct-mset* (*mset-nodes* ((*hd child*)) + *sum-list* (*map mset-nodes* (*tl child*)))›
‹*child* ≠ [] ⟹ *distinct-mset* (*mset-nodes* ((*hd child*)))›
using *distinct-mset-union* **by** (*cases child*; *auto*; *fail*)+
moreover have ‹*parents a* = *None*›
using *assms*
unfolding *encoded-hp-prop-list-def a*
by (*cases child*)
(*auto simp*: *ac-simps hp-parent-single-child-If3 hp-parent-simps-if*
dest: *multi-member-split*)
ultimately show *?thesis*
using *assms b*
unfolding *encoded-hp-prop-list-def a*
apply (*cases child*)
apply (*auto simp*: *ac-simps hp-parent-single-child-If3 hp-parent-simps-if*
dest: *multi-member-split*)
apply (*metis* (*no-types*, *lifting*) *disjunct-not-in distinct-mset-add insert-DiffM node-in-mset-nodes sum-mset.insert union-iff*)
apply (*metis hp-node-None-notin2 option.case-eq-if option.exhaust-sel*)
apply (*metis hp-node.simps*(*1*) *hp-node-children-simps2*)
apply (*metis hp-child.simps*(*1*) *hp-child-hp-children-simps2*)

done
qed

lemma *hp-parent-children-skip-first*[*simp*]:
‹$x \in\#$ *sum-list* (*map mset-nodes ch′*) $\Longrightarrow$
distinct-mset (*sum-list* (*map mset-nodes ch*) + *sum-list* (*map mset-nodes ch′*)) $\Longrightarrow$
hp-parent-children x (*ch* @ *ch′*) = *hp-parent-children x ch′*›
by (*induction ch*) (*auto simp*: *hp-parent-children-cons dest*!: *multi-member-split*)

lemma *hp-parent-children-skip-last*[*simp*]:
‹$x \in\#$ *sum-list* (*map mset-nodes ch*) $\Longrightarrow$
distinct-mset (*sum-list* (*map mset-nodes ch*) + *sum-list* (*map mset-nodes ch′*)) $\Longrightarrow$
hp-parent-children x (*ch* @ *ch′*) = *hp-parent-children x ch*›
by (*induction ch*) (*auto simp*: *hp-parent-children-cons dest*!: *multi-member-split split*: *option.splits*)

lemma *hp-parent-first-child*[*simp*]:
‹$n \neq m \Longrightarrow$ *hp-parent n* (*Hp m* w_m (*Hp n* w_n ch_n # ch_m)) = *Some* (*Hp m* w_m (*Hp n* w_n ch_n # ch_m))›
by (*auto simp*: *hp-parent.simps*(*1*))

lemma *encoded-hp-prop-list-link*:
fixes *m* ch_m *prevs b* hp_m *n nxts children parents*
defines ‹$prevs_0$ ≡ (*if* ch_m = [] *then prevs else prevs* (*node* (*hd* ch_m) := *Some n*))›
defines ‹*prevs′* ≡ (*if b* = [] *then* $prevs_0$ *else* $prevs_0$ (*node* (*hd b*) := *Some m*)) (*n*:= *None*)›
defines ‹*nxts′* ≡ *nxts* (*m* := *map-option node* (*option-hd b*), *n* := *map-option node* (*option-hd* ch_m))›
defines ‹*children′* ≡ *children* (*m* := *Some n*)›
defines ‹*parents′* ≡ (*if* ch_m = [] *then parents else parents*(*node* (*hd* ch_m) := *None*))(*n* := *Some m*)›
assumes ‹*encoded-hp-prop-list* $\mathcal{V}$ (*xs*) (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*) (*prevs*, *nxts*, *children*, *parents*, *scores*)›
shows ‹*encoded-hp-prop-list* $\mathcal{V}$ *xs* (*a* @ [*Hp m* w_m (*Hp n* w_n ch_n # ch_m)] @ *b*)
(*prevs′*, *nxts′*, *children′*, *parents′*, *scores*)›
proof −
have *dist*: ‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + (*sum-list* (*map mset-nodes* ch_n) +
($\sum_{\#}$ (*mset-nodes* ‘# *xs*) + (*sum-list* (*map mset-nodes a*) + *sum-list* (*map mset-nodes b*)))))›
and *notin*:
‹$n \notin\#$ *sum-list* (*map mset-nodes* ch_m)›
‹$n \notin\#$ *sum-list* (*map mset-nodes* ch_n)›
‹$n \notin\#$ *sum-list* (*map mset-nodes a*)›
‹$n \notin\#$ *sum-list* (*map mset-nodes b*)›
‹$m \notin\#$ *sum-list* (*map mset-nodes* ch_m)›
‹$m \notin\#$ *sum-list* (*map mset-nodes* ch_n)›
‹$m \notin\#$ *sum-list* (*map mset-nodes a*)›
‹$m \notin\#$ *sum-list* (*map mset-nodes b*)›
‹$n \neq m$› ‹$m \neq n$› **and**
nxts1: ‹(∀ *m′*∈#*xs*. ∀ *x*∈#*mset-nodes m′*. *nxts x* = *map-option node* (*hp-next x m′*))› **and**
prevs1: ‹(∀ *m*∈#*xs*. ∀ *x*∈#*mset-nodes m*. *prevs x* = *map-option node* (*hp-prev x m*))› **and**
parents1: ‹(∀ *m*∈#*xs*. ∀ *x*∈#*mset-nodes m*. *parents x* = *map-option node* (*hp-parent x m*))› **and**
child1: ‹(∀ *m*∈#*xs*. ∀ *x*∈#*mset-nodes m*. *children x* = *map-option node* (*hp-child x m*))› **and**
scores1: ‹(∀ *m*∈#*xs*. ∀ *x*∈#*mset-nodes m*. *scores x* = *map-option score* (*hp-node x m*))› **and**
nxts2: ‹(∀ *x*∈#$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).
nxts x = *map-option node* (*hp-next-children x* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
prevs2: ‹(∀ *x*∈#$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).
prevs x = *map-option node* (*hp-prev-children x* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
parents2: ‹(∀ *x*∈#$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).
parents x = *map-option node* (*hp-parent-children x* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
child2: ‹(∀ *x*∈#$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).

children x = map-option node (hp-child-children x (a @ [Hp m w_m ch_m, Hp n w_n ch_n] @ b)))› **and**
scores2: ‹(∀ x∈#∑# (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).
scores x = map-option score (hp-node-children x (a @ [Hp m w_m ch_m, Hp n w_n ch_n] @ b)))› **and**
dist2: ‹*distinct-mset* (∑# (*mset-nodes* ‘# *xs* + *mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
others-empty: ‹*empty-outside* (∑# (*mset-nodes* ‘# *xs* + *mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*))) *prevs*›
‹*empty-outside* (∑# (*mset-nodes* ‘# *xs* + *mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*))) *parents*› **and**
incl: ‹*set-mset* (∑# (*mset-nodes* ‘# *xs* + *mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*))) ⊆ *set-mset* $\mathcal{V}$›
using *assms* **unfolding** *encoded-hp-prop-list-def* **by** *auto*
have [*simp*]: ‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes* ch_m))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*) + *sum-list* (*map mset-nodes* ch_m))›
distinct-mset (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*)))
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_m) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*))))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes* ch_m)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + *sum-list* (*map mset-nodes* ch_m))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + *sum-list* (*map mset-nodes* ch_n))›
‹*distinct-mset* (*sum-list* (*map mset-nodes b*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes* ch_n))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes a*)))›
distinct-mset (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*))
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes* ch_n)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*)))›
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis distinct-mset-add union-ac*(*3*))
using *dist* **apply** (*smt* (*verit*, *del-insts*) *WB-List-More.distinct-mset-union2 group-cancel.add1 group-cancel.add2*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*smt* (*verit*, *del-insts*) *WB-List-More.distinct-mset-union2 group-cancel.add1 group-cancel.add2*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)

using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
done
have [*simp*]: ‹$m \neq$ *node* (*hd* ch_m)› ‹$n \neq$ *node* (*hd* ch_m)› ‹(*node* (*hd* ch_m)) $\notin\#$ *sum-list* (*map mset-nodes* *b*)›
‹*node* (*hd* ch_m) $\notin\#$ *sum-list* (*map mset-nodes* ch_n)›**if** ‹$ch_m \neq$ []›
using *dist that notin* **by** (*cases* ch_m; *auto dest*: *multi-member-split*; *fail*)+
have [*simp*]: ‹$m \neq$ *node* (*hd b*)› ‹$n \neq$ *node* (*hd b*)› **if** ‹$b \neq$ []›
using *dist that notin* **unfolding** *encoded-hp-prop-list-def* **by** (*cases b*; *auto*; *fail*)+
have [*simp*]: ‹*ma* $\in\#$ *xs* $\Longrightarrow$ *node* (*hd* ch_m) $\notin\#$ *mset-nodes ma*› **if** ‹$ch_m \neq$ []› **for** *ma*
using *dist that notin* **by** (*cases* ch_m; *auto dest*!: *multi-member-split*; *fail*)+
have [*simp*]: ‹*hp-parent-children* (*node* (*hd* ch_m)) ch_m = *None*›
by (*metis* ‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + *sum-list* (*map mset-nodes* ch_m))› *distinct-mset-add*
hp-parent.simps(*2*) *hp-parent-None-iff-children-None hp-parent-children-hd-None sum-image-mset-sum-map*)
define *NOTIN* **where**
‹*NOTIN x* ch_n ≡ *x* $\notin\#$ *sum-list* (*map mset-nodes* ch_n)› **for** *x* **and** ch_n :: ‹(′*a*, ′*b*) *hp list*›
have *K*[*unfolded NOTIN-def*[*symmetric*]]: ‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_n) $\Longrightarrow$ *x* $\notin\#$ *sum-list* (*map mset-nodes a*)›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_n) $\Longrightarrow$ *x* $\notin\#$ *sum-list* (*map mset-nodes b*)›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_n) $\Longrightarrow$ *x* $\notin\#$ *sum-list* (*map mset-nodes* ch_m)›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_n) $\Longrightarrow$ ¬*x* $\notin\#$ *sum-list* (*map mset-nodes* ch_n)›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_n) $\Longrightarrow$ *x* $\neq$ *m*›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_n) $\Longrightarrow$ *x* $\neq$ *n*›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_m) $\Longrightarrow$ *NOTIN x a*›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_m) $\Longrightarrow$ *NOTIN x b*›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_m) $\Longrightarrow$ *x* $\neq$ *m*›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_m) $\Longrightarrow$ *x* $\neq$ *n*›
‹*x* $\in\#$ *sum-list* (*map mset-nodes* ch_m) $\Longrightarrow$ *x* $\notin\#$ *sum-list* (*map mset-nodes* ch_n)› **and**
K′[*unfolded NOTIN-def*[*symmetric*]]:
‹*x* $\in\#$ *sum-list* (*map mset-nodes a*) $\Longrightarrow$ *x* $\notin\#$ *sum-list* (*map mset-nodes* ch_m)›
‹*x* $\in\#$ *sum-list* (*map mset-nodes a*) $\Longrightarrow$ *x* $\notin\#$ *sum-list* (*map mset-nodes* ch_n)›
‹*x* $\in\#$ *sum-list* (*map mset-nodes a*) $\Longrightarrow$ *x* $\notin\#$ *sum-list* (*map mset-nodes b*)›
‹*x* $\in\#$ *sum-list* (*map mset-nodes a*) $\Longrightarrow$ *x* $\neq$ *m*›
‹*x* $\in\#$ *sum-list* (*map mset-nodes a*) $\Longrightarrow$ *x* $\neq$ *n*› **and**
K″[*unfolded NOTIN-def*[*symmetric*]]:
‹*x* $\in\#$ *sum-list* (*map mset-nodes b*) $\Longrightarrow$ (*x* $\notin\#$ *sum-list* (*map mset-nodes a*))›
‹*x* $\in\#$ *sum-list* (*map mset-nodes b*) $\Longrightarrow$ *x* $\notin\#$ *sum-list* (*map mset-nodes* ch_n)›
‹*x* $\in\#$ *sum-list* (*map mset-nodes b*) $\Longrightarrow$ *x* $\neq$ *m*›
‹*x* $\in\#$ *sum-list* (*map mset-nodes b*) $\Longrightarrow$ *x* $\neq$ *n*›
for *x*
using *dist notin* **by** (*auto dest*!: *multi-member-split simp*: *NOTIN-def*)

note [*simp*] = *NOTIN-def*[*symmetric*]
show *?thesis*
using *dist2* **unfolding** *encoded-hp-prop-list-def prod.simps assms*(*1*−*5*)
apply (*intro conjI impI allI*)
subgoal using *assms* **unfolding** *encoded-hp-prop-list-def*
by (*auto simp*: *ac-simps simp del*: *NOTIN-def*[*symmetric*])
subgoal using *incl* **by** *auto*
subgoal using *nxts1*
by *auto*
subgoal using *prevs1*
apply (*cases* ch_m; *cases b*)
apply (*auto*)

apply (*metis WB-List-More.distinct-mset-union2 add-diff-cancel-right′ distinct-mem-diff-mset mset-add node-in-mset-nodes sum-mset.insert union-iff*)
apply (*metis* (*no-types*, *lifting*) *add-mset-disjoint*(*1*) *distinct-mset-add mset-add node-in-mset-nodes sum-mset.insert union-iff*)+
done
subgoal
using *child1*
by *auto*
subgoal
using *parents1*
by *auto*
subgoal
using *scores1*
by *auto*
subgoal
using *nxts2*
by (*auto dest*: *multi-member-split simp*: *K hp-next-children-append-single-remove-children*)
subgoal
using *prevs2* **supply** [*cong del*] = *image-mset-cong*
by (*auto simp add*: *K K′ K″ hp-prev-children-append-single-remove-children hp-prev-skip-hd-children map-option-skip-in-child*)
subgoal
using *child2 notin*(*9*)
by (*auto simp add*: *K K′ K″ hp-child-children-skip-first*[*of* - ‹[-]›, *simplified*]
hp-child-children-skip-first[*of* - ‹- # -›, *simplified*]
hp-child-children-skip-last[*of* - - ‹[-]›, *simplified*]
hp-child-children-skip-last[*of* - ‹[-]›, *simplified*] *notin*
hp-child-children-skip-last[*of* - ‹[-, -]›, *simplified*]
hp-child-children-skip-first[*of* - - ‹[-]›, *simplified*]
split: *option.splits*)
subgoal
using *parents2 notin*(*9*)
by (*auto simp add*: *K K′ K″ hp-parent-children-skip-first*[*of* - ‹[-]›, *simplified*]
hp-parent-children-skip-first[*of* - ‹- # -›, *simplified*] *hp-parent-simps-single-if*
hp-parent-children-skip-last[*of* - - ‹[-]›, *simplified*]
hp-parent-children-skip-last[*of* - ‹[-]›, *simplified*] *notin*
hp-parent-children-skip-last[*of* - ‹[-, -]›, *simplified*]
hp-parent-children-skip-first[*of* - - ‹[-]›, *simplified*]
split: *option.splits*)
subgoal
using *scores2*
by (*auto split*: *option.splits simp*: *K K′ K″ hp-node-children-Cons-if*
ex-hp-node-children-Some-in-mset-nodes
dest: *multi-member-split*)
subgoal
using *others-empty*
by (*auto simp add*: *K K′ K″ ac-simps*)
subgoal
using *others-empty*
by (*auto simp add*: *K K′ K″ ac-simps*)
done
qed

lemma *encoded-hp-prop-list-link2*:
fixes *m* ch_m *prevs b* hp_m *n nxts children* ch_n *a parents*
defines ‹*prevs′* ≡ (*if* ch_n = [] *then prevs else prevs* (*node* (*hd* ch_n) := *Some m*))(*m* := *None*, *n* :=

map-option node (*option-last a*))›
defines ‹$nxts_0$ ≡ (*if a* = [] *then nxts else nxts*(*node* (*last a*) := *Some n*))›
defines ‹*nxts′* ≡ $nxts_0$ (*n* := *map-option node* (*option-hd b*), *m* := *map-option node* (*option-hd* ch_n))›
defines ‹*children′* ≡ *children* (*n* := *Some m*)›
defines ‹*parents′* ≡ (*if* ch_n = [] *then parents else parents*(*node* (*hd* ch_n) := *None*))(*m* := *Some n*)›
assumes ‹*encoded-hp-prop-list* $\mathcal{V}$ (*xs*) (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*) (*prevs*, *nxts*, *children*, *parents*, *scores*)›
shows ‹*encoded-hp-prop-list* $\mathcal{V}$ *xs* (*a* @ [*Hp n* w_n (*Hp m* w_m ch_m # ch_n)] @ *b*)
(*prevs′*, *nxts′*, *children′*, *parents′*, *scores*)›
proof −
have *dist*: ‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + (*sum-list* (*map mset-nodes* ch_n) +
($\sum_{\#}$ (*mset-nodes* ‘# *xs*) + (*sum-list* (*map mset-nodes a*) + *sum-list* (*map mset-nodes b*))))))›
and *notin*:
‹*n* ∉# *sum-list* (*map mset-nodes* ch_m)›
‹*n* ∉# *sum-list* (*map mset-nodes* ch_n)›
‹*n* ∉# *sum-list* (*map mset-nodes a*)›
‹*n* ∉# *sum-list* (*map mset-nodes b*)›
‹*m* ∉# *sum-list* (*map mset-nodes* ch_m)›
‹*m* ∉# *sum-list* (*map mset-nodes* ch_n)›
‹*m* ∉# *sum-list* (*map mset-nodes a*)›
‹*m* ∉# *sum-list* (*map mset-nodes b*)›
‹*n* ≠ *m*› ‹*m* ≠ *n*› **and**
nxts1: ‹(∀ *m′*∈#*xs*. ∀ *x*∈#*mset-nodes m′*. *nxts x* = *map-option node* (*hp-next x m′*))› **and**
prevs1: ‹(∀ *m*∈#*xs*. ∀ *x*∈#*mset-nodes m*. *prevs x* = *map-option node* (*hp-prev x m*))› **and**
child1: ‹(∀ *m*∈#*xs*. ∀ *x*∈#*mset-nodes m*. *children x* = *map-option node* (*hp-child x m*))› **and**
parent1: ‹(∀ *m*∈#*xs*. ∀ *x*∈#*mset-nodes m*. *parents x* = *map-option node* (*hp-parent x m*))› **and**
nxts2: ‹(∀ *x*∈#$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).
nxts x = *map-option node* (*hp-next-children x* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
prevs2: ‹(∀ *x*∈#$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).
prevs x = *map-option node* (*hp-prev-children x* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
child2: ‹(∀ *x*∈#$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).
children x = *map-option node* (*hp-child-children x* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
parent2: ‹(∀ *x*∈#$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).
parents x = *map-option node* (*hp-parent-children x* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
scores2: ‹(∀ *x*∈#$\sum_{\#}$ (*mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)).
scores x = *map-option score* (*hp-node-children x* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
scores1: ‹(∀ *m*∈#*xs*. ∀ *x*∈#*mset-nodes m*. *scores x* = *map-option score* (*hp-node x m*))› **and**
dist2: ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *xs* + *mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*)))› **and**
others-empty: ‹*empty-outside* ($\sum_{\#}$ (*mset-nodes* ‘# *xs* + *mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*))) *prevs*›
‹*empty-outside* ($\sum_{\#}$ (*mset-nodes* ‘# *xs* + *mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*))) *parents*› **and**
incl: ‹*set-mset* ($\sum_{\#}$ (*mset-nodes* ‘# *xs* + *mset-nodes* ‘# *mset* (*a* @ [*Hp m* w_m ch_m, *Hp n* w_n ch_n] @ *b*))) ⊆ *set-mset* $\mathcal{V}$›
using *assms* **unfolding** *encoded-hp-prop-list-def prod.simps* **by** *clarsimp-all*
have [*simp*]: ‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes* ch_m))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*) + *sum-list* (*map mset-nodes* ch_m))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_m) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*))))›

‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes* ch_m)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + *sum-list* (*map mset-nodes* ch_m))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + *sum-list* (*map mset-nodes* ch_n))›
‹*distinct-mset* (*sum-list* (*map mset-nodes b*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes* ch_n))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes* ch_m))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + (*sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes a*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes* ch_n)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_n) + (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes b*) + (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes* ch_n))))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes* ch_n) + *sum-list* (*map mset-nodes b*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*))›
‹*distinct-mset* (*sum-list* (*map mset-nodes a*) + (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*)))›
‹*distinct-mset* (*sum-list* (*map mset-nodes* ch_m) + *sum-list* (*map mset-nodes b*))›
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis distinct-mset-add union-ac*(*3*))
using *dist* **apply** (*smt* (*verit*, *del-insts*) *WB-List-More.distinct-mset-union2 group-cancel.add1 group-cancel.add2*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*smt* (*verit*, *del-insts*) *WB-List-More.distinct-mset-union2 group-cancel.add1 group-cancel.add2*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*smt* (*verit*, *del-insts*) *WB-List-More.distinct-mset-union2 union-commute union-lcomm*)
using *dist* **apply** (*smt* (*verit*, *del-insts*) *WB-List-More.distinct-mset-union2 union-commute union-lcomm*)
using *dist* **apply** (*smt* (*verit*, *del-insts*) *WB-List-More.distinct-mset-union2 union-commute union-lcomm*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
using *dist* **apply** (*metis* (*no-types*, *lifting*) *distinct-mset-add union-assoc union-commute*)
done
have [*simp*]: ‹$m \neq$ *node* (*hd* ch_m)› ‹$n \neq$ *node* (*hd* ch_m)› ‹(*node* (*hd* ch_m)) $\notin\#$ *sum-list* (*map mset-nodes b*)›

‹node (hd ch_m) ∉# sum-list (map mset-nodes ch_n)›**if** ‹ch_m ≠ []›
using *dist that notin* **by** (*cases* ch_m; *auto dest*: *multi-member-split*; *fail*)+
have [*simp*]: ‹m ≠ node (hd ch_n)› ‹n ≠ node (hd ch_n)› ‹(node (hd ch_n)) ∉# sum-list (map mset-nodes b)›
‹node (hd ch_n) ∉# sum-list (map mset-nodes ch_m)›**if** ‹ch_n ≠ []›
using *dist that notin* **by** (*cases* ch_n; *auto dest*: *multi-member-split*; *fail*)+
have [*simp*]: ‹m ≠ node (hd b)› ‹n ≠ node (hd b)› **if** ‹b ≠ []›
using *dist that notin* **unfolding** *encoded-hp-prop-list-def* **by** (*cases b*; *auto*; *fail*)+

define *NOTIN* **where**
‹NOTIN x ch_n ≡ x ∉# sum-list (map mset-nodes ch_n)› **for** *x* **and** ch_n :: ‹('a, 'b) hp list›
have *K*[*unfolded NOTIN-def*[*symmetric*]]: ‹x ∈# sum-list (map mset-nodes ch_n) ⟹ x ∉# sum-list (map mset-nodes a)›
‹x ∈# sum-list (map mset-nodes ch_n) ⟹ x ∉# sum-list (map mset-nodes b)›
‹x ∈# sum-list (map mset-nodes ch_n) ⟹ x ∉# sum-list (map mset-nodes ch_m)›
‹x ∈# sum-list (map mset-nodes ch_n) ⟹ x ≠ m›
‹x ∈# sum-list (map mset-nodes ch_n) ⟹ x ≠ n›
‹x ∈# sum-list (map mset-nodes ch_m) ⟹ NOTIN x a›
‹x ∈# sum-list (map mset-nodes ch_m) ⟹ NOTIN x b›
‹x ∈# sum-list (map mset-nodes ch_m) ⟹ x ≠ m›
‹x ∈# sum-list (map mset-nodes ch_m) ⟹ x ≠ n›
‹x ∈# sum-list (map mset-nodes ch_m) ⟹ x ∉# sum-list (map mset-nodes ch_n)› **and**
K'[*unfolded NOTIN-def*[*symmetric*]]:
‹x ∈# sum-list (map mset-nodes a) ⟹ x ∉# sum-list (map mset-nodes ch_m)›
‹x ∈# sum-list (map mset-nodes a) ⟹ x ∉# sum-list (map mset-nodes ch_n)›
‹x ∈# sum-list (map mset-nodes a) ⟹ x ∉# sum-list (map mset-nodes b)›
‹x ∈# sum-list (map mset-nodes a) ⟹ x ≠ m›
‹x ∈# sum-list (map mset-nodes a) ⟹ x ≠ n› **and**
K''[*unfolded NOTIN-def*[*symmetric*]]:
‹x ∈# sum-list (map mset-nodes b) ⟹ (x ∉# sum-list (map mset-nodes a))›
‹x ∈# sum-list (map mset-nodes b) ⟹ x ∉# sum-list (map mset-nodes ch_n)›
‹x ∈# sum-list (map mset-nodes b) ⟹ x ∉# sum-list (map mset-nodes ch_m)›
‹x ∈# sum-list (map mset-nodes b) ⟹ x ≠ m›
‹x ∈# sum-list (map mset-nodes b) ⟹ x ≠ n›
for *x*
using *dist notin* **by** (*auto dest*!: *multi-member-split simp*: *NOTIN-def*)
have [*simp*]: ‹node (last a) ∉# sum-list (map mset-nodes ch_m)›
‹node (last a) ∉# sum-list (map mset-nodes ch_n)› **if** ‹a ≠ []›
using *that dist* **by** (*cases a rule*: *rev-cases*; *cases* ‹last a›; *auto*; *fail*)+
note [*simp*] = *NOTIN-def*[*symmetric*]

have [*simp*]: ‹hp-parent-children (node (hd ch_m)) ch_m = None›
by (*metis* ‹distinct-mset (sum-list (map mset-nodes a) + sum-list (map mset-nodes ch_m))› *distinct-mset-add*
hp-parent.simps(*2*) *hp-parent-None-iff-children-None hp-parent-children-hd-None sum-image-mset-sum-map*)
have [*simp*]: ‹ch_n ≠ [] ⟹ hp-parent-children (node (hd ch_n)) ch_n = None›
using *dist*
by (*cases* ch_n; *cases* ‹hd ch_n›) (*auto simp*: *hp-parent-children-cons distinct-mset-add split*: *option.splits*)
have [*iff*]: ‹ch_n ≠ [] ⟹ ma ∈# xs ⟹ node (hd ch_n) ∈# mset-nodes ma ⟷ False› **for** *ma*
using *dist2* **apply** *auto*
by (*metis* (*no-types*, *lifting*) *add-mset-disjoint*(*1*) *distinct-mset-add insert-DiffM inter-mset-empty-distrib-right node-hd-in-sum sum-mset.insert*)
show *?thesis*
using *dist2* **unfolding** *encoded-hp-prop-list-def prod.simps assms*(*1−5*)
apply (*intro conjI impI allI*)

subgoal using *assms* **unfolding** *encoded-hp-prop-list-def*
by (*auto simp*: *ac-simps simp del*: *NOTIN-def*[*symmetric*])
subgoal using *incl* **by** *auto*
subgoal using *nxts1*
apply (*cases a rule*: *rev-cases*)
apply *auto*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-right′ distinct-mset-in-diff mset-add node-in-mset-nodes sum-mset.insert union-iff*)
subgoal using *prevs1*
by *auto*
subgoal
using *child1*
by *auto*
subgoal
using *parent1*
by *auto*
subgoal
using *scores1*
by *auto*
subgoal
using *nxts2*
by (*auto dest*: *multi-member-split simp*: *K hp-next-children-append-single-remove-children*
hp-next-children-skip-last-not-last
notin)
subgoal
using *prevs2* **supply** [*cong del*] = *image-mset-cong*
by (*auto simp add*: *K K′ K″ hp-prev-children-append-single-remove-children hp-prev-skip-hd-children map-option-skip-in-child hp-prev-children-skip-first-append*[*of* - ‹[-]›, *simplified*])
subgoal
using *child2*
by (*auto simp add*: *K K′ K″ hp-child-children-skip-first*[*of* - ‹[-]›, *simplified*]
hp-child-children-skip-first[*of* - ‹- # -›, *simplified*]
hp-child-children-skip-last[*of* - - ‹[-]›, *simplified*]
hp-child-children-skip-last[*of* - ‹[-]›, *simplified*] *notin*
hp-child-children-skip-last[*of* - ‹[-, -]›, *simplified*]
hp-child-children-skip-first[*of* - - ‹[-,-]›, *simplified*]
hp-child-children-skip-first[*of* - - ‹[-]›, *simplified*]
split: *option.splits*)
subgoal
using *parent2 notin*(*9*)
by (*auto simp add*: *K K′ K″ hp-parent-children-skip-first*[*of* - ‹[-]›, *simplified*]
hp-parent-children-skip-first[*of* - ‹- # -›, *simplified*] *hp-parent-simps-single-if*
hp-parent-children-skip-last[*of* - - ‹[-]›, *simplified*]
hp-parent-children-skip-last[*of* - ‹[-]›, *simplified*] *notin*
hp-parent-children-skip-last[*of* - ‹[-, -]›, *simplified*]
hp-parent-children-skip-first[*of* - - ‹[-]›, *simplified*]
hp-parent-children-skip-first[*of* - - ‹[-,-]›, *simplified*]
eq-commute[*of n* ‹*node* (*hd* [])›]
split: *option.splits*)
subgoal
using *scores2*
by (*auto split*: *option.splits simp*: *K K′ K″ hp-node-children-Cons-if*
ex-hp-node-children-Some-in-mset-nodes
dest: *multi-member-split*)
subgoal
using *others-empty*

by (*auto simp add*: *K K′ K″ ac-simps add-mset-commute*[*of m n*])
subgoal
using *others-empty*
by (*auto simp add*: *K K′ K″ ac-simps add-mset-commute*[*of m n*])
done
qed

definition *encoded-hp-prop-list-conc*
:: *′a*::*linorder multiset* × (*′a*, *′b*) *hp-fun* × *′a option* ⇒
′a multiset × (*′a*, *′b*) *hp option* ⇒ *bool*
where
‹*encoded-hp-prop-list-conc* = (λ(𝒱, *arr*, *h*) (𝒱′, *x*). 𝒱 = 𝒱′ ∧
(*case x of None* ⇒ *encoded-hp-prop-list* 𝒱′ {#} ([]:: (*′a*, *′b*) *hp list*) *arr* ∧ *h* = *None*
| *Some x* ⇒ *encoded-hp-prop-list* 𝒱′ {#*x*#} [] *arr* ∧ *set-mset* (*mset-nodes x*) ⊆ *set-mset* 𝒱 ∧ *h* = *Some* (*node x*)))›

lemma *encoded-hp-prop-list-conc-alt-def*:
‹*encoded-hp-prop-list-conc* = (λ(𝒱, *arr*, *h*) (𝒱′, *x*). 𝒱 = 𝒱′ ∧
(*case x of None* ⇒ *encoded-hp-prop-list* 𝒱′ {#} ([]:: (*′a*::*linorder*, *′b*) *hp list*) *arr* ∧ *h* = *None*
| *Some x* ⇒ *encoded-hp-prop-list* 𝒱′ {#*x*#} [] *arr* ∧ *h* = *Some* (*node x*)))›
unfolding *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def*
by (*auto split*: *option.splits intro*!: *ext*)

definition *encoded-hp-prop-list2-conc*
:: *′a*::*linorder multiset* × (*′a*, *′b*) *hp-fun* × *′a option* ⇒
′a multiset × (*′a*, *′b*) *hp list* ⇒ *bool*
where
‹*encoded-hp-prop-list2-conc* = (λ(𝒱, *arr*, *h*) (𝒱′, *x*). 𝒱′ = 𝒱 ∧
(*encoded-hp-prop-list* 𝒱 {#} *x arr* ∧ *set-mset* (*sum-list* (*map mset-nodes x*)) ⊆ *set-mset* 𝒱 ∧ *h* = *None*))›

lemma *encoded-hp-prop-list2-conc-alt-def*:
‹*encoded-hp-prop-list2-conc* = (λ(𝒱, *arr*, *h*) (𝒱′, *x*). 𝒱 = 𝒱′ ∧
(*encoded-hp-prop-list* 𝒱 {#} *x arr* ∧ *h* = *None*))›
unfolding *encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def*
by (*auto split*: *option.splits intro*!: *ext*)

lemma *encoded-hp-prop-list-update-score*:
fixes *h* :: ‹(*′a*, *nat*) *hp*› **and** *a arr* **and** *hs* :: ‹(*′a*, *nat*) *hp multiset*› **and** *x*
defines *arr′*: ‹*arr′* ≡ *hp-update-score a* (*Some x*) *arr*›
assumes *enc*: ‹*encoded-hp-prop-list* 𝒱 (*add-mset* (*Hp a b c*) *hs*) [] *arr*›
shows ‹*encoded-hp-prop-list* 𝒱 (*add-mset* (*Hp a x c*) *hs*) []
arr′›
proof −
obtain *prevs nxts childs parents scores* 𝒱 **where**
arr: ‹*arr* = ((*prevs*, *nxts*, *childs*, *parents*, *scores*))› **and**
dist: ‹*distinct-mset* (*sum-list* (*map mset-nodes c*) + ∑# (*mset-nodes* ‘# *hs*))›
‹*a* ∉# *sum-list* (*map mset-nodes c*)›
‹*a* ∉# ∑# (*mset-nodes* ‘# *hs*)›
and
𝒱: ‹*set-mset* (*sum-list* (*map mset-nodes xs*)) ⊆ 𝒱›
by (*cases arr*) (*use assms* **in** ‹*auto simp*: *ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def encoded-hp-prop-def*›)
have *find-key-in-nodes*: ‹*find-key a h* ≠ *None* ⟹ *node* (*the* (*find-key a h*)) ∈# *mset-nodes h*›
by (*cases* ‹*a* ∈# *mset-nodes h*›)
(*use find-key-None-or-itself*[*of a h*] **in** ‹*auto simp del*: *find-key-None-or-itself*›)

have *in-find-key-in-nodes1*: ‹*x* ∈# *mset-nodes y* ⟹ *find-key a h* = *Some y* ⟹ *x* ∈# *mset-nodes h*›
for *x y*
using *mset-nodes-find-key-subset*[*of a h*]
by *auto*
have [*simp*]: ‹*find-key a h* = *None* ⟹ *remove-key a h* = *Some h*›
by (*metis find-key.simps find-key-none-iff hp.exhaust-sel hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-simps2 option-last-Nil option-last-Some-iff* (*2*) *remove-key-notin-unchanged*)
have [*simp*]: ‹*map-option node* (*hp-parent xa* (*Hp a b c*)) = *map-option node* (*hp-parent xa* (*Hp a x c*))› **for** *xa*
by (*cases c*; *auto simp*: *hp-parent.simps*(*1*))
have ‹*remove-key a h* ≠ *None* ⟹ *node* (*the* (*remove-key a h*)) ∈# *mset-nodes h*›
by (*metis remove-key.simps get-min2.simps hp.exhaust-sel option.collapse option.distinct*(*2*) *remove-key-notin-unchanged*)
then show *?thesis*
supply [[*goals-limit=1*]]
using *enc*
unfolding *arr hp-update-nxt-def hp-update-prev-def case-prod-beta encoded-hp-prop-list-def prod.simps arr′* **apply** −
apply (*intro conjI impI ballI*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
apply (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
by (*metis hp-child-hp-children-simps2*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
subgoal
by (*auto simp*: *find-remove-mset-nodes-full hp-update-score-def*)
done
qed

Refinement to Imperative version

definition *hp-insert* :: ‹′*a* ⇒ ′*b*::*linorder* ⇒ ′*a multiset* × (′*a*,′*b*) *hp-fun* × ′*a option* ⇒ (′*a multiset* × (′*a*,′*b*) *hp-fun* × ′*a option*) *nres*› **where**

```
‹hp-insert = (λ(i::'a) (w::'b) (𝒱::'a multiset, arr :: ('a, 'b) hp-fun, h :: 'a option). do {
if h = None then do {
  ASSERT (i ∈# 𝒱);
  RETURN (𝒱, hp-set-all i None None None None (Some w) arr, Some i)
 } else do {
  ASSERT (i ∈# 𝒱);
  ASSERT (hp-read-prev i arr = None);
  ASSERT (hp-read-parent i arr = None);
  let (j::'a) = ((the h) :: 'a);
  ASSERT (j ∈# 𝒱 ∧ j ≠ i);
  ASSERT (hp-read-score j (arr :: ('a, 'b) hp-fun) ≠ None);
  ASSERT (hp-read-prev j arr = None ∧ hp-read-nxt j arr = None ∧ hp-read-parent j arr = None);
  let y = (the (hp-read-score j arr)::'b);
  if y < w
  then do {
    let arr = hp-set-all i None None (Some j) None (Some (w::'b)) (arr::('a, 'b) hp-fun);
    let arr = hp-update-parents j (Some i) arr;
    let nxt = hp-read-nxt j arr;
    RETURN (𝒱, arr :: ('a, 'b) hp-fun, Some i)
  }
  else do {
    let child = hp-read-child j arr;
    ASSERT (child ≠ None ⟶ the child ∈# 𝒱);
    let arr = hp-set-all j None None (Some i) None (Some y) arr;
    let arr = hp-set-all i None child None (Some j) (Some (w::'b)) arr;
    let arr = (if child = None then arr else hp-update-prev (the child) (Some i) arr);
    let arr = (if child = None then arr else hp-update-parents (the child) None arr);
    RETURN (𝒱, arr :: ('a, 'b) hp-fun, h)
  }
 }
})›
```

lemma *hp-insert-spec*:
  assumes ‹*encoded-hp-prop-list-conc arr h*› **and**
    ‹*snd h ≠ None ⟹ i ∉# mset-nodes (the (snd h))*› **and**
    ‹*i ∈# fst arr*›
  shows ‹*hp-insert i w arr ≤ ⇓ {(arr, h). encoded-hp-prop-list-conc arr h} (ACIDS.mop-hm-insert i w h)*›
proof −
  let *?h* = ‹*snd h*›
  obtain *prevs nxts childs scores parents 𝒱* **where**
    arr: ‹*arr = (𝒱, (prevs, nxts, childs, parents, scores), map-option node ?h)*›
   by (*cases arr*; *cases ?h*) (*use assms* **in** ‹*auto simp: encoded-hp-prop-list-conc-def encoded-hp-prop-list-def encoded-hp-prop-def*›)

  have *enc*: ‹*encoded-hp-prop 𝒱 {#Hp i w [the ?h]#}*
    (prevs, nxts(i := None, node (the ?h) := None), childs(i ↦ node (the ?h)), parents(node (the ?h) ↦ i), scores(i ↦ w))› **and**
    enc2: ‹*encoded-hp-prop 𝒱 {#Hp (node (the ?h)) (score (the ?h)) (Hp i w [] # hps (the ?h))#}*
   (if hps (the ?h) = [] then prevs else prevs(node (hd (hps (the ?h))) ↦ node (Hp i w [])),
    nxts (i := None, node (Hp i w []) := if hps (the ?h) = [] then None else Some (node (hd (hps (the ?h))))),
    childs(i := None)(node (the ?h) ↦ node (Hp i w [])),
    (if hps (the ?h) = [] then parents else parents(node (hd (hps (the ?h))) := None))(node (Hp i w []) ↦ node (the ?h)),

scores(*i* ↦ *w*, *node* (*the ?h*) ↦ *score* (*the ?h*)))› (**is** *?G*)
if ‹*?h* ≠ *None*›
proof −
have ‹*encoded-hp-prop* 𝒱 {#*the ?h*#} (*prevs*, *nxts*, *childs*, *parents*, *scores*)›
using *that assms* **by** (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def encoded-hp-prop-def arr*)
then have *0*: ‹*encoded-hp-prop* 𝒱 {#*Hp i w* [], *the ?h*#}
(*prevs*, *nxts*(*i* := *None*), *childs*(*i* := *None*), *parents*, *scores*(*i* ↦ *w*))›
using *encoded-hp-prop-irrelevant*[*of i* ‹{#*the ?h*#}› 𝒱 *prevs nxts childs parents scores w*] *that assms*
by (*auto simp*: *arr*)
from *encoded-hp-prop-link*[*OF this*]
show ‹*encoded-hp-prop* 𝒱 {#*Hp i w* [*the ?h*]#}
(*prevs*, *nxts*(*i* := *None*, *node* (*the ?h*) := *None*), *childs*(*i* ↦ *node* (*the ?h*)), *parents*(*node* (*the ?h*) ↦ *i*), *scores*(*i* ↦ *w*))›
by *auto*
from *0* **have** ‹*encoded-hp-prop* 𝒱 {#*Hp* (*node* (*the ?h*)) (*score* (*the ?h*)) (*hps* (*the ?h*)), *Hp i w* []#}
(*prevs*, *nxts*(*i* := *None*), *childs*(*i* := *None*), *parents*, *scores*(*i* ↦ *w*))›
by (*cases* ‹*the ?h*›) (*auto simp*: *add-mset-commute*)
from *encoded-hp-prop-link*[*OF this*]
show *?G* **.**
qed
have *prev-parent-i*:
‹*?h* ≠ *None* ⟹ *hp-read-prev i* (*prevs*, *nxts*, *childs*, *parents*, *scores*) = *None*›
‹*?h* ≠ *None* ⟹ *hp-read-parent i* (*prevs*, *nxts*, *childs*, *parents*, *scores*) = *None*›
using *assms* **unfolding** *encoded-hp-prop-list-conc-def*
by (*force simp*: *arr encoded-hp-prop-def empty-outside-alt-def dest*!: *multi-member-split*[*of i*])+
have *1*: ‹*?h* ≠ *None* ⟹ *hps* (*the ?h*) ≠ [] ⟹ *i* ≠ *node* (*hd* (*hps* (*the ?h*)))›
using *assms* **by** (*cases* ‹*the ?h*›; *cases* ‹*hps* (*the ?h*)›; *cases h*) *auto*
have [*simp*]: ‹*encoded-hp-prop* 𝒱 {#*Hp x1a x2 x3*#} (*prevs*, *nxts*, *childs*, *parents*, *scores*) ⟹ *scores x1a* = *Some x2*›
‹*encoded-hp-prop* 𝒱 {#*Hp x1a x2 x3*#} (*prevs*, *nxts*, *childs*, *parents*, *scores*) ⟹ *parents x1a* = *None*›
‹*encoded-hp-prop* 𝒱 {#*Hp x1a x2 x3*#} (*prevs*, *nxts*, *childs*, *parents*, *scores*) ⟹ *childs x1a* = *map-option node* (*option-hd x3*)› **for** *x1a x2 x3*
by (*auto simp*: *encoded-hp-prop-def*)
show *?thesis*
using *assms*
unfolding *hp-insert-def arr prod.simps ACIDS.mop-hm-insert-def*
apply *refine-vcg*
subgoal
by *auto*
subgoal
by (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def hp-set-all-def empty-outside-alt-def*
split: *option.splits prod.splits*)
subgoal
by *auto*
subgoal using *prev-parent-i* **by** *auto*
subgoal using *prev-parent-i* **by** *auto*
subgoal
by (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def hp-set-all-def*
split: *option.splits prod.splits*)
subgoal
by (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def hp-set-all-def*
split: *option.splits prod.splits*)
subgoal
by (*cases* ‹*the ?h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def hp-set-all-def*

split: *option.splits prod.splits*)
subgoal
by (*cases* ‹*the ?h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def hp-set-all-def* *split*: *option.splits prod.splits*)
subgoal
by (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def hp-set-all-def* *split*: *option.splits prod.splits*)
subgoal
by (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def hp-set-all-def* *split*: *option.splits prod.splits*)
subgoal
using *enc*
by (*cases h*, *simp*; *cases* ‹*the ?h*›)
(*auto simp*: *hp-set-all-def encoded-hp-prop-list-conc-def fun-upd-idem hp-update-parents-def*)
subgoal
using *enc*
by (*cases h*, *simp*; *cases* ‹*the ?h*›; *cases* ‹*hps* (*the ?h*)›; *cases* ‹*hd* (*hps* (*the ?h*))›)
(*auto simp*: *hp-set-all-def encoded-hp-prop-list-conc-def fun-upd-idem hp-update-parents-def* *arr*)
subgoal
using *enc2 1*
by (*cases h*, *simp*; *cases* ‹*the ?h*›)
(*auto simp*: *hp-set-all-def encoded-hp-prop-list-conc-def fun-upd-idem hp-update-prev-def* *fun-upd-twist hp-update-parents-def*)
done
qed

definition *hp-link* :: ‹$'a \Rightarrow 'a \Rightarrow 'a$ *multiset* $\times$ ($'a,'b$::*order*) *hp-fun* $\times$ $'a$ *option* $\Rightarrow$ (($'a$ *multiset* $\times$ ($'a,'b$) *hp-fun* $\times$ $'a$ *option*) $\times$ $'a$) *nres*› **where**
‹*hp-link* = (λ(i::$'a$) j ($\mathcal{V}$::$'a$ *multiset*, *arr* :: ($'a$, $'b$) *hp-fun*, h :: $'a$ *option*). *do* {
ASSERT ($i \neq j$);
ASSERT ($i \in\# \mathcal{V}$);
ASSERT ($j \in\# \mathcal{V}$);
ASSERT (*hp-read-score i arr* $\neq$ *None*);
ASSERT (*hp-read-score j arr* $\neq$ *None*);
let x = (*the* (*hp-read-score i arr*)::$'b$);
let y = (*the* (*hp-read-score j arr*)::$'b$);
let prev = *hp-read-prev i arr*;
let nxt = *hp-read-nxt j arr*;
ASSERT (*nxt* $\neq$ *Some i* $\wedge$ *nxt* $\neq$ *Some j*);
ASSERT (*prev* $\neq$ *Some i* $\wedge$ *prev* $\neq$ *Some j*);
let (*parent*,*ch*,w_p, w_{ch}) = (*if* $y < x$ *then* (i, j, x, y) *else* (j, i, y, x));
let child = *hp-read-child parent arr*;
ASSERT (*child* $\neq$ *Some i* $\wedge$ *child* $\neq$ *Some j*);
let $child_{ch}$ = *hp-read-child ch arr*;
ASSERT ($child_{ch}$ $\neq$ *Some i* $\wedge$ $child_{ch}$ $\neq$ *Some j* $\wedge$ ($child_{ch}$ $\neq$ *None* $\longrightarrow$ $child_{ch}$ $\neq$ *child*));
ASSERT (*distinct* ([i, j] @ (*if* $child_{ch}$ $\neq$ *None then* [*the* $child_{ch}$] *else* [])
@ (*if child* $\neq$ *None then* [*the child*] *else* [])
@ (*if prev* $\neq$ *None then* [*the prev*] *else* [])
@ (*if nxt* $\neq$ *None then* [*the nxt*] *else* []))
);
ASSERT (*ch* $\in\# \mathcal{V}$);
ASSERT (*parent* $\in\# \mathcal{V}$);
ASSERT (*child* $\neq$ *None* $\longrightarrow$ *the child* $\in\# \mathcal{V}$);
ASSERT (*nxt* $\neq$ *None* $\longrightarrow$ *the nxt* $\in\# \mathcal{V}$);
ASSERT (*prev* $\neq$ *None* $\longrightarrow$ *the prev* $\in\# \mathcal{V}$);

let arr = hp-set-all parent prev nxt (Some ch) None (Some (w_p::'b)) (arr::('a, 'b) hp-fun);
let arr = hp-set-all ch None child $child_{ch}$ (Some parent) (Some (w_{ch}::'b)) (arr::('a, 'b) hp-fun);
let arr = (if child = None then arr else hp-update-prev (the child) (Some ch) arr);
let arr = (if nxt = None then arr else hp-update-prev (the nxt) (Some parent) arr);
let arr = (if prev = None then arr else hp-update-nxt (the prev) (Some parent) arr);
let arr = (if child = None then arr else hp-update-parents (the child) None arr);
RETURN ((𝒱, arr :: ('a, 'b) hp-fun, h), parent)
})›

lemma *fun-upd-twist2*: $a \neq c \Longrightarrow a \neq e \Longrightarrow c \neq e \Longrightarrow m(a := b,\ c := d,\ e := f) = (m(e := f,\ c := d))(a := b)$
by *auto*

lemma *hp-link*:
assumes *enc*: ‹*encoded-hp-prop-list2-conc arr* (𝒱', *xs* @ *x* # *y* # *ys*)› **and**
‹*i* = *node x*› **and**
‹*j* = *node y*›
shows ‹*hp-link i j arr* ≤ *SPEC* (λ(*arr*, *n*). *encoded-hp-prop-list2-conc arr* (𝒱', *xs* @ *ACIDS.link x y* # *ys*) ∧
n = *node* (*ACIDS.link x y*))›
proof −
obtain *prevs nxts childs parents scores* 𝒱 **where**
arr: ‹*arr* = (𝒱, (*prevs*, *nxts*, *childs*, *parents*, *scores*), *None*)› **and**
dist: ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* '# (*mset* (*xs* @ *x* # *y* # *ys*))))› **and**
𝒱: ‹*set-mset* (*sum-list* (*map mset-nodes* (*xs* @ *x* # *y* # *ys*))) ⊆ *set-mset* 𝒱› **and**
𝒱'[*simp*]: ‹𝒱' = 𝒱›
by (*cases arr*) (*use assms* **in** ‹*auto simp*: *ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def encoded-hp-prop-def*›)

have *ij*: ‹*i* ≠ *j*›
using *dist assms*(*2*,*3*) **by** (*cases x*; *cases y*) *auto*
have *xy*: ‹*Hp* (*node x*) (*score x*) (*hps x*) = *x*› ‹*Hp* (*node y*) (*score y*) (*hps y*) = *y*› **and**
sc: ‹*score x* = *the* (*scores i*)› ‹*score y* = *the* (*scores j*)› **and**
link-x-y: ‹*ACIDS.link x y* = *ACIDS.link* (*Hp i* (*the* (*scores i*)) (*hps x*))
(*Hp j* (*the* (*scores j*)) (*hps y*))›
by (*cases x*; *cases y*; *use assms* **in** ‹*auto simp*: *encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def arr*
simp del: *ACIDS.link.simps*›; *fail*)+
obtain ch_x w_x ch_y w_y **where**
x: ‹$x = Hp\ i\ w_x\ ch_x$› **and**
y: ‹$y = Hp\ j\ w_y\ ch_y$›
using *assms*(*2*−*3*)
by (*cases y*; *cases x*) *auto*
have *sc'*:
‹*scores i* = *Some* w_x›
‹*scores j* = *Some* w_y›
‹*prevs i* = *map-option node* (*option-last xs*)›
‹*nxts i* = *Some j*›
‹*prevs j* = *Some i*›
‹*nxts j* = *map-option node* (*option-hd ys*)›
‹*childs i* = *map-option node* (*option-hd* ch_x)›
‹*childs j* = *map-option node* (*option-hd* ch_y)›
‹*xs* ≠ [] ⟹ *nxts* (*node* (*last xs*)) = *Some i*›
‹*ys* ≠ [] ⟹ *prevs* (*node* (*hd ys*)) = *Some j*›
‹*parents i* = *None*›

‹parents j = None›
using *assms*(*1*) *x y* **apply** (*auto simp*: *ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def encoded-hp-prop-def arr hp-child-children-Cons-if*)
apply (*smt* (*verit*) *assms*(*3*) *hp-next-None-notin-children hp-next-children.elims list.discI list.inject list.sel*(*1*) *option-hd-Nil option-hd-Some-hd*)
using *assms*(*1*) *x y* **apply** (*cases xs rule*: *rev-cases*; *auto simp*: *ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def*
encoded-hp-prop-def arr)
apply (*metis WB-List-More.distinct-mset-union2 add-diff-cancel-right′ assms*(*2*) *distinct-mset-in-diff hp-next-children-simps*(*1*) *hp-next-children-skip-first-append node-in-mset-nodes option.map*(*2*) *sum-image-mset-sum-map*)
using *assms*(*1*) *x y* **apply** (*cases ys*;*auto simp*: *ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def encoded-hp-prop-def arr*)
apply (*cases* ‹*hd ys*›)
apply (*auto simp*:)
by (*simp add*: *hp-parent-None-notin-same-hd hp-parent-children-cons*)

have *par*: ‹*parents j* = *None*› ‹*parents i* = *None*›
‹$ch_x \neq [] \Longrightarrow$ *parents* (*node* (*hd* ch_x)) = *Some i*›
‹$ch_y \neq [] \Longrightarrow$ *parents* (*node* (*hd* ch_y)) = *Some j*›
using *assms*(*1*) *x y* **apply** (*auto simp*: *ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def encoded-hp-prop-def arr hp-child-children-Cons-if*)
apply (*metis hp-parent-None-iff-children-None hp-parent-None-notin-same-hd hp-parent-children-cons hp-parent-hd-None sum-image-mset-sum-map*)
apply (*metis assms*(*2*) *hp-parent-children-cons hp-parent-simps-single-if option.simps*(*5*))
apply (*cases* ch_y)
apply (*simp-all add*: *hp-parent-children-skip-first*[*of - -* ‹[-]›, *simplified*] *distinct-mset-add*)
apply (*subst hp-parent-children-skip-first*[*of - -* ‹[-]›, *simplified*])
apply *simp*
apply *simp*
apply (*metis distinct-mset-add inter-mset-empty-distrib-right*)
apply (*subst hp-parent-children-skip-last*[*of -* ‹[-]›, *simplified*])
apply *simp*
apply *simp*
apply (*metis distinct-mset-add inter-mset-empty-distrib-right*)
apply *simp*
done

have *diff*:
‹*nxts j* ≠ *Some j*› ‹*nxts j* ≠ *Some i*› ‹*nxts i* ≠ *Some i*›
‹*prevs i* ≠ *Some i*› ‹*prevs i* ≠ *Some j*›
‹*childs i* ≠ *Some i*› ‹*childs i* ≠ *Some j*›
‹*childs j* ≠ *Some i*› ‹*childs j* ≠ *Some j*› ‹*childs i* ≠ *None* ⟹ *childs i* ≠ *childs j*›
‹*childs j* ≠ *None* ⟹ *childs i* ≠ *childs j*›
‹*prevs i* ≠ *None* ⟹ *prevs i* ≠ *nxts j*›
using *dist sc′* **unfolding** *x y* **apply** (*cases ys*; *cases xs rule*: *rev-cases*; *auto split*: *if-splits*; *fail*)+
using *dist sc′* **unfolding** *x y* **apply** (*cases ys*; *cases xs rule*: *rev-cases*; *cases* ‹*last xs*›; *cases* ‹*hd ys*›; *cases* ch_x; *cases* ch_y; *cases* ‹*hd* ch_x›; *cases* ‹*hd* ch_y›; *auto split*: *if-splits*; *fail*)+
done
have *dist2*:
‹*distinct*([*i*, *j*]
@ (*if childs i* ≠ *None then* [*the* (*childs i*)] *else* [])
@ (*if childs j* ≠ *None then* [*the* (*childs j*)] *else* [])
@ (*if prevs i* ≠ *None then* [*the* (*prevs i*)] *else* [])
@ (*if nxts j* ≠ *None then* [*the* (*nxts j*)] *else* []))›
using *dist sc′* **unfolding** *x y* **by** (*cases ys*; *cases xs rule*: *rev-cases*; *cases* ‹*last xs*›; *cases* ‹*hd ys*›;

cases ch_x; *cases* ch_y; *cases* ‹*hd* ch_x›; *cases* ‹*hd* ch_y›)
(*auto split*: *if-splits*)
have *if-pair*: ‹(*if a then* (*b*, *c*) *else* (*d*,*f*)) = (*if a then b else d*, *if a then c else f*)› **for** *a b c d f*
by *auto*
have *enc0*: ‹*encoded-hp-prop-list* $\mathcal{V}$ {#} (*xs* @ [*Hp* (*node x*) (*score x*) (*hps x*), *Hp* (*node y*) (*score y*) (*hps y*)] @ *ys*) (*prevs*, *nxts*, *childs*, *parents*, *scores*)›
using *enc* **unfolding** *x y* **by** (*auto simp*: *encoded-hp-prop-list2-conc-def arr*)
then have *H*: ‹*fst x1*= $\mathcal{V}$ $\Longrightarrow$ *snd* (*snd x1*) = *None*$\Longrightarrow$ *encoded-hp-prop-list2-conc x1* ($\mathcal{V}'$, *xs* @ *ACIDS.link x y* # *ys*) $\longleftrightarrow$
encoded-hp-prop-list $\mathcal{V}$ {#} (*xs* @ *ACIDS.link x y* # *ys*) (*fst* (*snd x1*))› **for** *x1*
using *dist* $\mathcal{V}$ **unfolding** *x y*
by (*cases x1*)
(*simp add*: *encoded-hp-prop-list2-conc-def*)
have *KK* [*intro*!]: ‹ch_x ≠ [] $\Longrightarrow$ *ys* ≠ [] $\Longrightarrow$ *node* (*hd* ch_x) ≠ *node* (*hd ys*)›
using *dist2 sc'* **by** *simp*

have *subs*: ‹*set-mset* (*sum-list* (*map mset-nodes* (*xs* @ *Hp i* w_x ch_x # *Hp j* w_y ch_y # *ys*))) ⊆ *set-mset* $\mathcal{V}$›
using *assms*(*1*) *sc'*(*7*,*3*) **unfolding** *encoded-hp-prop-list2-conc-def x y arr prod.simps*
encoded-hp-prop-list-def
by (*clarsimp-all simp*: *encoded-hp-prop-list-def*)
then have *childs-i*: ‹*childs i* ≠ *None* $\Longrightarrow$ *the* (*childs i*) ∈# $\mathcal{V}$›
‹*prevs i* ≠ *None* $\Longrightarrow$ *the* (*prevs i*) ∈# $\mathcal{V}$›
using *sc'*(*7*,*3*) **unfolding** *encoded-hp-prop-list2-conc-def x y arr prod.simps*
encoded-hp-prop-list-def
apply (*clarsimp-all simp*: *encoded-hp-prop-list-def*)
apply (*metis node-hd-in-sum option.sel subsetD*)
by (*metis* $\mathcal{V}$ *dist distinct-mset-add hp-next-children-None-notin hp-next-children-last*
list.discI map-append option-hd-Some-hd option-last-Nil option-last-Some-iff(*2*)
subset-eq sum-image-mset-sum-map sum-list-append)
have *childs-j*: ‹*childs j* ≠ *None* $\Longrightarrow$ *the* (*childs j*) ∈# $\mathcal{V}$›
‹*nxts j* ≠ *None* $\Longrightarrow$ *the* (*nxts j*) ∈# $\mathcal{V}$›
using *subs sc'*(*6*,*8*) **unfolding** *encoded-hp-prop-list2-conc-def x y arr prod.simps*
apply (*clarsimp-all simp*: *encoded-hp-prop-list-def*)
apply (*metis node-hd-in-sum option.sel subsetD*)
by (*metis basic-trans-rules*(*31*) *node-hd-in-sum option.sel*)
show *?thesis*
unfolding *hp-link-def arr prod.simps*
apply *refine-vcg*
subgoal using *ij* **by** *auto*
subgoal using *dist* $\mathcal{V}$ **by** (*auto simp*: *x*)
subgoal using *dist* $\mathcal{V}$ **by** (*auto simp*: *y*)
subgoal using *sc'* **by** *auto*
subgoal using *sc'* **by** *auto*
subgoal using *diff* **by** *auto*
subgoal using *diff* **by** *auto*
subgoal using *diff* **by** (*auto split*: *if-splits*)
subgoal using *diff* **by** (*auto split*: *if-splits*)
subgoal using *diff* **by** (*auto split*: *if-splits*)
subgoal using *diff* **by** (*auto split*: *if-splits*)
subgoal using *diff* **by** (*auto split*: *if-splits*)
subgoal using *diff* **by** (*auto split*: *if-splits*)
subgoal using *diff* **by** (*auto split*: *if-splits*)
subgoal using *dist2* **by** (*clarsimp split*: *if-splits*)
subgoal by (*clarsimp split*: *if-splits*)
subgoal by (*clarsimp split*: *if-splits*)

subgoal using *childs-i childs-j* **by** (*clarsimp simp*: *split*: *if-splits*)
subgoal using *childs-i childs-j* **by** (*clarsimp simp*: *split*: *if-splits*)
subgoal using *childs-i childs-j* **by** (*clarsimp simp*: *split*: *if-splits*)
subgoal premises *p* **for** *parent b ch ba* w_p w_c *x1 x2*
apply (*cases* ‹*the* (*scores j*) < *the* (*scores i*)›)
subgoal
apply (*subst H*)
using *p*(*1*−*12*) *p*(*19*)[*symmetric*] *dist2* $\mathcal{V}$
apply (*solves simp*)
using *p*(*1*−*12*) *p*(*19*)[*symmetric*] *dist2* $\mathcal{V}$
apply (*solves simp*)
apply (*subst arg-cong2*[*THEN iffD1*, *of* - - - - ‹*encoded-hp-prop-list* $\mathcal{V}$ {#}›, *OF* - - *encoded-hp-prop-list-link*[*of* $\mathcal{V}$ ‹{#}› *xs* ‹*node x*› ‹*score x*› ‹*hps x*› ‹*node y*› ‹*score y*› ‹*hps y*› *ys*
prevs nxts childs parents scores, *OF enc0*]])
subgoal
using *sc′ p*(*1*−*12*) *p*(*19*)[*symmetric*] *dist2* $\mathcal{V}$
by (*simp add*: *x y*)
subgoal
using *sc′ p*(*1*−*12*) *p*(*19*)[*symmetric*] *dist2* $\mathcal{V}$ *par*
apply (*simp add*: *x y*)
apply (*intro conjI impI*)
subgoal apply (*simp add*: *fun-upd-idem fun-upd-twist fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›] *hp-set-all-def*)
apply (*subst fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›])
apply *auto*[*2*]
done
subgoal
supply [[*goals-limit*=*1*]]
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist*
hp-update-prev-def
hp-update-parents-def)
apply (*subst fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*subst fun-upd-twist*[*of* - - *parents*])
apply *force*
apply (*subst fun-upd-twist*[*of* - - *prevs*])
apply *force*
apply *blast*
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist*
hp-update-prev-def)
apply (*subst fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*subst fun-upd-idem*[*of* ‹*nxts*(*parent* := *None*)›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*simp* (*no-asm-simp*))
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist*
hp-update-prev-def hp-update-parents-def)
apply (*subst fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›])
apply (*simp* (*no-asm-simp*))

apply *force*
apply (*subst fun-upd-twist*[*of - - prevs*])
apply *force*
apply (*subst fun-upd-twist*[*of - - parents*])
apply *force*
apply (*simp* (*no-asm-simp*))
apply (*subst fun-upd-idem*[*of* ‹*nxts*(*parent* := *None*)(*ch* ↦ *node* (*hd* ch_x))›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply *force*
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*subst fun-upd-twist*[*of - - prevs*])
apply *force*
apply (*simp* (*no-asm-simp*))
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*subst fun-upd-twist2*[*of - - - prevs*])
apply (*rule KK*)
apply *assumption*
apply *assumption*
apply *force*
apply *force*
apply (*subst fun-upd-twist*[*of - -* ‹*prevs*(*ch* := *None*)›])
apply (*rule KK*[*symmetric*])
apply *assumption*
apply *assumption*
apply (*subst fun-upd-twist*[*of - -* ‹*parents*›])
apply *argo*
apply *blast*
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*subst fun-upd-twist2*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-twist*[*of - - prevs*])
apply *force*
apply (*subst fun-upd-idem*[*of* ‹*nxts*› ‹*node* (*last xs*)›])
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*simp* (*no-asm*))

apply (*subst fun-upd-twist*[*of* - - ‹*nxts*›])
apply *argo*
apply *blast*
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*subst fun-upd-twist2*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-twist*[*of* - - ‹*prevs*(*ch* := *None*)›])
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-idem*[*of* ‹*nxts*(*ch* ↦ *node* (*hd* ch_x), *parent* ↦ *node* (*hd ys*))›])
apply (*simp* (*no-asm-simp*))
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*simp* (*no-asm-simp*))
apply (*subst fun-upd-twist*[*of* - - ‹*parents*›])
apply *argo*
apply *blast*
done
done
apply (*rule TrueI*)
done
subgoal
supply [[*goals-limit*=*1*]]
apply (*subst H*)
using *p*(*1*−*12*) *p*(*19*)[*symmetric*] *dist2* $\mathcal{V}$
apply *simp*
using *p*(*1*−*12*) *p*(*19*)[*symmetric*] *dist2* $\mathcal{V}$
apply *simp*
apply (*subst arg-cong2*[*THEN iffD1*, *of* - - - - ‹*encoded-hp-prop-list* $\mathcal{V}$ {#}›, *OF* - - *encoded-hp-prop-list-link2*[*of* $\mathcal{V}$ ‹{#}› *xs* ‹*node x*› ‹*score x*› ‹*hps x*› ‹*node y*› ‹*score y*› ‹*hps y*› *ys prevs nxts childs parents scores*, *OF enc0*]]])
subgoal
using *sc′ p*(*1*−*12*) *p*(*19*)[*symmetric*] *dist2* $\mathcal{V}$
by (*simp add*: *x y*)
subgoal
using *sc′ p*(*1*−*12*) *p*(*19*)[*symmetric*] *dist2* $\mathcal{V}$
apply (*simp add*: *x y*)
apply (*intro conjI impI*)
subgoal
by (*simp add*: *fun-upd-idem fun-upd-twist fun-upd-idem*[*of* ‹*childs*(*parent* ↦ *ch*)›] *hp-set-all-def*)
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-twist*[*of* - - *prevs*])
apply *force*
apply (*subst fun-upd-idem*[*of prevs* -])
apply (*simp* (*no-asm-simp*))
apply (*subst fun-upd-twist*[*of* - - *prevs*])
apply *force*
apply (*subst fun-upd-twist*[*of* - - *parents*])

apply *force*
apply (*simp* (*no-asm-simp*))
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def*)
apply (*subst fun-upd-twist*[*of - - prevs*])
apply *force*
apply (*subst fun-upd-twist2*[*of - - - nxts*])
apply *force*
apply *force*
apply *force*
apply (*subst fun-upd-idem*[*of* ‹*nxts*(*ch* := *None*)›])
apply (*simp* (*no-asm-simp*))
apply (*simp* (*no-asm-simp*))
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-idem*[*of* ‹(*nxts*(*node* (*last xs*) ↦ *parent*))›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*subst fun-upd-twist*[*of - - prevs*])
apply *force*
apply (*subst fun-upd-twist2*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-twist*[*of - - parents*])
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-twist*[*of - - nxts*])
apply *force*
apply (*subst fun-upd-twist*[*of - -* ‹(*prevs*(*parent* ↦ *node* (*last xs*)))›])
apply *force*
apply (*simp* (*no-asm-simp*))
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-idem*[*of* ‹*prevs*(*parent* := *None*)›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*simp* (*no-asm-simp*))
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-twist2*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-twist*[*of - - parents*])
apply *force*
apply (*simp* (*no-asm-simp*))
apply (*subst fun-upd-idem*[*of* ‹*prevs*(*parent* := *None*)›])
apply (*simp* (*no-asm-simp*))

apply (*subst fun-upd-idem*[*of* ‹(*prevs*(*parent* := *None*))(- ↦ -)›])
apply (*simp* (*no-asm-simp*))
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply *force*
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-twist*[*of* - - *prevs*])
apply *force*
apply (*subst fun-upd-idem*[*of* ‹(*prevs*(*parent* ↦ *node* (*last xs*)))(*ch* := *None*)›])
apply (*simp* (*no-asm-simp*))
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-idem*[*of* ‹*nxts*(*node* (*last xs*) ↦ *parent*)›])
apply (*simp* (*no-asm-simp*))
apply *force*
apply (*subst fun-upd-twist*[*of* - - *nxts*])
apply *force*
apply (*simp* (*no-asm-simp*))
done
subgoal
apply (*simp* (*no-asm-simp*) *add*: *hp-set-all-def hp-update-nxt-def fun-upd-idem fun-upd-twist hp-update-prev-def hp-update-parents-def*)
apply (*subst fun-upd-idem*[*of* ‹(*prevs*(*parent* ↦ *node* (*last xs*)))(*ch* := *None*)(*node* (*hd* ch_y) ↦ *ch*)›])
apply (*simp* (*no-asm-simp*))
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-twist2*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-idem*[*of* ‹*nxts*(*node* (*last xs*) ↦ *parent*)›])
apply (*simp* (*no-asm-simp*))
apply (*smt* (*z3*) *IntI Un-iff empty-iff mem-Collect-eq option.simps*(*9*) *option-hd-Some-hd*)
apply (*subst fun-upd-twist*[*of* - - *nxts*])
apply *force*
apply (*subst fun-upd-twist*[*of* - - *parents*])
apply *force*
apply (*simp* (*no-asm*))
done
done
apply (*rule TrueI*)
done
done
subgoal premises *p*
using *p*(*1*−*12*) *p*(*19*)[*symmetric*] *dist2* $\mathcal{V}$
using sc'
by (*cases* ‹*the* (*scores j*) < *the* (*scores i*)›) (*simp-all add*: *x y split*: *if-split*)
done
qed

In an imperative setting use the two pass approaches is better than the alternative.
The *e* of the loop is a dummy counter.

definition $vsids\text{-}pass_1$ **where**

‹vsids-pass$_1$ = (λ(𝒱::'a multiset, arr :: ('a, 'b::order) hp-fun, h :: 'a option) (j::'a). do {
((𝒱, arr, h), j, -, n) ← WHILE$_T$(λ((𝒱, arr, h), j, e, n). j ≠ None)
(λ((𝒱, arr, h), j, e::nat, n). do {
if j = None then RETURN ((𝒱, arr, h), None, e, n)
else do {
let j = the j;
ASSERT (j ∈# 𝒱);
let nxt = hp-read-nxt j arr;
if nxt = None then RETURN ((𝒱, arr, h), nxt, e+1, j)
else do {
ASSERT (nxt ≠ None);
ASSERT (the nxt ∈# 𝒱);
let nnxt = hp-read-nxt (the nxt) arr;
((𝒱, arr, h), n) ← hp-link j (the nxt) (𝒱, arr, h);
RETURN ((𝒱, arr, h), nnxt, e+2, n)
}}
})
((𝒱, arr, h), Some j, 0::nat, j);
RETURN ((𝒱, arr, h), n)
})›

lemma *vsids-pass$_1$*:
fixes *arr* :: ‹'a::linorder multiset × ('a, nat) hp-fun × 'a option›
assumes ‹encoded-hp-prop-list2-conc arr (𝒱', xs)› **and** ‹xs ≠ []› **and** ‹j = node (hd xs)›
shows ‹vsids-pass$_1$ arr j ≤ SPEC(λ(arr, j). encoded-hp-prop-list2-conc arr (𝒱', ACIDS.pass$_1$ xs) ∧ j = node (last (ACIDS.pass$_1$ xs)))›
proof −
obtain *prevs nxts childs scores 𝒱* **where**
arr: ‹arr = (𝒱, (prevs, nxts, childs, scores), None)› **and**
dist: ‹distinct-mset ($\sum_{\#}$ (mset-nodes '# (mset (xs))))› **and**
𝒱: ‹set-mset (sum-list (map mset-nodes xs)) ⊆ set-mset 𝒱› **and**
[*simp*]: ‹𝒱' = 𝒱›
by (*cases arr*) (*use assms* **in** ‹auto simp: ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def encoded-hp-prop-def›)
define *I* **where** ‹I ≡ (λ(arr, nnxt::'a option, e, k).
encoded-hp-prop-list2-conc arr (𝒱, ACIDS.pass$_1$(take e xs) @ drop e xs) ∧ nnxt = map-option node (option-hd (drop (e) xs)) ∧
e ≤ (length xs) ∧ (nnxt = None ⟷ e = length xs) ∧ (nnxt ≠ None ⟶ even e) ∧
k = (if e=0 then j else node (last (ACIDS.pass$_1$(take e xs)))))›
have *I0*: ‹I ((𝒱, (prevs, nxts, childs, scores), None), Some j, 0, j)›
using *assms* **unfolding** *I-def prod.simps*
by (*cases xs, auto simp: arr; fail*)+
have *I-no-next*: ‹I ((𝒱, arr, ch'), None, Suc e, y)›
if ‹I ((𝒱, arr, ch'), Some y, e, n)› **and**
‹hp-read-nxt y arr = None›
for *s a b prevs x2 nxts children x1b x2b x1c x2c x1d x2d arr e y ch' 𝒱 n*
proof −
have ‹e = length xs − 1› ‹xs ≠ []›
using *that*
apply (*cases* ‹drop e xs›; *cases* ‹hd (drop e xs)›)
apply (*auto simp*: *I-def encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def*)
apply (*subst* (*asm*) *hp-next-children-hd-simps*)
apply *simp*
apply *simp*
apply (*rule distinct-mset-mono*)

prefer *2*
apply *assumption*
apply (*auto simp*: *drop-is-single-iff*)
using *that*
apply (*auto simp*: *I-def*)
done
then show *?thesis*
using *that ACIDS.pass*$_1$*-append-even*[*of* ‹*butlast xs*› ‹[*last xs*]›]
by (*auto simp*: *I-def*)
qed

have *link-pre1*: ‹*encoded-hp-prop-list2-conc* (*x1*, *x1a*, *x2a*)
($\mathcal{V}'$, *ACIDS.pass*$_1$ (*take x2b xs*) @
xs!*x2b* # *xs*!(*Suc x2b*) # *drop* (*x2b*+*2*) *xs*)› (**is** *?H1*) **and**
link-pre2: ‹*the x1b* = *node* (*xs* ! *x2b*)› (**is** *?H2*) **and**
link-pre3: ‹*the* (*hp-read-nxt* (*the x1b*) *x1a*) = *node* (*xs* ! *Suc x2b*)› (**is** *?H3*)
if ‹*I s*› **and**
s: ‹*case s of* (*x*, *xa*) ⇒ (*case x of* ($\mathcal{V}$, *arr*, *h*) ⇒ λ(*j*, *e*, *n*). *j* ≠ *None*) *xa*›
‹*s* = (*a*, *b*)›
x2b′ = (*x2b*, *j*)
‹*b* = (*x1b*, *x2b*′)›
‹*x2* = (*x1a*, *x2a*)›
‹*a* = (*x1*, *x2*)›
‹*x1b* ≠ *None*› **and**
nxt: ‹*hp-read-nxt* (*the x1b*) *x1a* ≠ *None*›
for *s a b x1 x2 x1a x2a x1b x2b j x2b*′
proof −
have ‹*encoded-hp-prop-list x1* {#} (*ACIDS.pass*$_1$ (*take x2b xs*) @ *drop x2b xs*) *x1a*›
‹*x2b* < *length xs*›
‹*x1b* = *Some* (*node* (*hd* (*drop x2b xs*)))›
using *that*
by (*auto simp*: *I-def encoded-hp-prop-list2-conc-def arr*)
then have ‹*drop x2b xs* ≠ []› ‹*tl* (*drop x2b xs*) ≠ []› ‹*Suc x2b* < *length xs*› ‹*the x1b* = *node* (*xs* ! *x2b*)›
‹*the* (*hp-read-nxt* (*the x1b*) *x1a*) = *node* (*xs* ! *Suc x2b*)›
using *nxt* **unfolding** *s* **apply** −
apply (*cases* ‹*drop x2b xs*›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
apply (*cases* ‹*drop x2b xs*›; *cases* ‹*hd* (*drop x2b xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
apply (*cases* ‹*drop x2b xs*›; *cases* ‹*hd* (*drop x2b xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
apply (*smt* (*verit*) *Suc-le-eq append-eq-conv-conj hp-next-None-notin-children*
hp-next-children.elims length-Suc-conv-rev list.discI list.inject nat-less-le
option-last-Nil option-last-Some-iff(*2*))
apply (*cases* ‹*drop x2b xs*›; *cases* ‹*hd* (*drop x2b xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
apply (*subst* (*asm*) *hp-next-children-hd-simps*)
apply *simp*
apply *simp*
apply (*rule distinct-mset-mono*′)
apply *assumption*
apply (*auto simp*: *drop-is-single-iff*)
apply (*metis hd-drop-conv-nth hp.sel*(*1*) *list.sel*(*1*))
apply (*cases* ‹*drop x2b xs*›; *cases* ‹*tl* (*drop x2b xs*)›; *cases* ‹*hd* (*drop x2b xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)

by (*metis Cons-nth-drop-Suc list.inject nth-via-drop*)
then show *?H1*
using *that* ‹*x2b* < *length xs*›
by (*cases* ‹*drop x2b xs*›; *cases* ‹*tl* (*drop x2b xs*)›)
(*auto simp*: *I-def encoded-hp-prop-list2-conc-def Cons-nth-drop-Suc*)
show *?H2 ?H3* **using** ‹*the x1b* = *node* (*xs* ! *x2b*)›
‹*the* (*hp-read-nxt* (*the x1b*) *x1a*) = *node* (*xs* ! *Suc x2b*)› **by** *fast+*
qed
have *I-Suc-Suc*: ‹*I* ((*x2c*, *x2d*, *xe*), *hp-read-nxt* (*the* (*hp-read-nxt* (*the nxt*) *x2a*)) *x2a*, *k* + *2*, *n*)›
if
inv: ‹*I s*› **and**
brk: ‹*case s of* (*x*, *xa*) ⇒ (*case x of* ($\mathcal{V}$, *arr*, *h*) ⇒ λ(*j*, *e*, *n*). *j* ≠ *None*) *xa*› **and**
st: ‹*s* = (*arr2*, *b*)›
‹*b* = (*nxt*, *k*′)›
‹*k*′ = (*k*, *j*)›
‹*x1a* = (*x2a*, *x1b*)›
‹*arr2* = ($\mathcal{V}''$, *x1a*)›
‹*linkedn* = (*linked*, *n*)›
‹*x1d* = (*x2d*, *xe*)›
‹*linked* = (*x2c*, *x1d*)› **and**
nxt: ‹*nxt* ≠ *None*› **and**
nxts: ‹*hp-read-nxt* (*the nxt*) *x2a* ≠ *None*›
‹*hp-read-nxt* (*the nxt*) *x2a* ≠ *None*› **and**
linkedn: ‹*case linkedn of*
(*arr*, *n*) ⇒
encoded-hp-prop-list2-conc arr
($\mathcal{V}'$, *ACIDS.pass*$_1$ (*take k xs*) @ *ACIDS.link* (*xs* ! *k*) (*xs* ! *Suc k*) # *drop* (*k* + *2*) *xs*) ∧
n = *node* (*ACIDS.link* (*xs* ! *k*) (*xs* ! *Suc k*))›
for *s arr2 b x1a x2a x1b nxt k linkedn linked n x2c x1d x2d xe j k*′ $\mathcal{V}''$
proof −
have *enc*: ‹*encoded-hp-prop-list* $\mathcal{V}'$ {#} (*ACIDS.pass*$_1$ (*take k xs*) @ *drop k xs*) *x2a*›
‹*k* < *length xs*›
‹*nxt* = *Some* (*node* (*hd* (*drop k xs*)))› **and**
dist: ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# (*mset* (*ACIDS.pass*$_1$ (*take k xs*) @ *drop k xs*))))›
using *that*
by (*auto simp*: *I-def encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def*)

then have ‹*drop k xs* ≠ []› ‹*tl* (*drop k xs*) ≠ []› ‹*Suc k* < *length xs*› ‹*the nxt* = *node* (*xs* ! *k*)›
using *nxt* **unfolding** *st* **apply** −
apply (*cases* ‹*drop k xs*›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
apply (*cases* ‹*drop k xs*›; *cases* ‹*hd* (*drop k xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
apply (*cases* ‹*drop k xs*›; *cases* ‹*hd* (*drop k xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
apply (*metis hp-read-nxt.simps option.sel that*(*12*))
apply (*cases* ‹*drop k xs*›; *cases* ‹*hd* (*drop k xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
apply (*subst* (*asm*) *hp-next-children-hd-simps*)
apply *simp*
apply *simp*
apply (*rule distinct-mset-mono*′)
apply *assumption*
apply (*auto simp*: *drop-is-single-iff*)
apply (*metis Some-to-the Suc-lessI drop-eq-ConsD drop-eq-Nil2 hp-read-nxt.simps nat-in-between-eq*(*1*) *option.map*(*1*) *option-hd-Nil that*(*12*))

apply (*cases* ‹*drop k xs*›; *cases* ‹*tl* (*drop k xs*)›; *cases* ‹*hd* (*drop k xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
apply (*metis hp.sel*(*1*) *nth-via-drop*)
by (*metis hp.sel*(*1*) *nth-via-drop*)
then have *le*: ‹*Suc* (*Suc k*) $\leq$ *length xs*›
using *enc nxts* **unfolding** *st nxt* **apply** −
apply (*cases* ‹*drop k xs*›; *cases* ‹*tl* (*drop k xs*)›; *cases* ‹*hd* (*drop k xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def*)
done
have *take-nth*: ‹*take* (*Suc* (*Suc k*)) *xs* = *take k xs* @ [*xs*!*k*, *xs*!*Suc k*]›
using *le* **by** (*auto simp*: *take-Suc-conv-app-nth*)
have *nnxts*: ‹*hp-read-nxt* (*the* (*hp-read-nxt* (*node* (*hd* (*drop k xs*))) *x2a*)) *x2a* =
map-option node (*option-hd* (*drop* (*Suc* (*Suc k*)) *xs*))›
using *enc nxts le* ‹*tl* (*drop k xs*) $\neq$ []› **unfolding** *st nxt* **apply** −
apply (*cases* ‹*drop k xs*›; *cases* ‹*tl* (*drop k xs*)›; *cases* ‹*hd* (*tl* (*drop k xs*))›; *cases* ‹*hd* (*drop k xs*)›)
apply (*auto simp*: *I-def encoded-hp-prop-list-def arr*)
apply (*subst hp-next-children-hd-simps*)
apply (*solves simp*)
apply (*rule distinct-mset-mono*′[*OF dist*])
by (*auto simp*: *drop-is-single-iff drop-Suc-nth*)
show *?thesis*
using *inv nxt le linkedn nnxts*
unfolding *st*
by (*auto simp*: *I-def take-Suc take-nth ACIDS.pass*$_1$*-append-even*)
qed

show *?thesis*
unfolding *vsids-pass*$_1$*-def arr prod.simps*
apply (*refine-vcg WHILET-rule*[**where** *I*=*I* **and** *R* = ‹*measure* (λ(*arr*, *nnxt*::′*a option*, *e*, -). *length xs* −*e*)›]
hp-link)
subgoal by *auto*
subgoal by (*rule I0*)
subgoal by (*auto simp*: *I-def*)
subgoal by (*auto simp*: *I-def*)
subgoal by (*auto simp*: *I-def encoded-hp-prop-list2-conc-def*)
subgoal for *s a b x1 x2 x1a x2a x1b x2b*
by (*auto simp*: *I-no-next*)
subgoal by (*auto simp*: *I-def*)
subgoal for *s a b x1 x2 x1a x2a x1b x2b x1c x2c*
using *hp-next-children-in-nodes2*[*of* ‹(*node* (*hd* (*drop x1c xs*)))› ‹(*ACIDS.pass*$_1$ (*take x1c xs*) @ *drop x1c xs*)›]
by (*auto 5 3 simp*: *I-def encoded-hp-prop-list-def encoded-hp-prop-list2-conc-def*)
apply (*rule link-pre1*; *assumption?*)
apply (*rule link-pre2*; *assumption*)
subgoal premises *p* **for** *s a b x1 x2 x1a x2a x1b x2b*
using *link-pre3*[*OF p*(*1*−*8*)] *p*(*9*−)
by *auto*
subgoal for *s arr2 b* $\mathcal{V}'$ *x1a x2a x1b nxt k linkedn linked n x2c x1d x2d xe*
by (*rule I-Suc-Suc*)
subgoal
by (*auto simp*: *I-def*)
subgoal
by (*auto simp*: *I-def*)
subgoal
using *assms*

by (*auto simp*: *I-def*)
done
qed

definition *vsids-pass*$_2$ **where**
‹*vsids-pass*$_2$ = (λ(𝒱::′*a multiset*, *arr* :: (′*a*, ′*b*::*order*) *hp-fun*, *h* :: ′*a option*) (*j*::′*a*). *do* {
ASSERT (*j* ∈# 𝒱);
let nxt = *hp-read-prev j arr*;
((𝒱, *arr*, *h*), *j*, *leader*, -) ← *WHILE*$_T$(λ((𝒱, *arr*, *h*), *j*, *leader*, *e*). *j* ≠ *None*)
(λ((𝒱, *arr*, *h*), *j*, *leader*, *e*::*nat*). *do* {
if j = *None then RETURN* ((𝒱, *arr*, *h*), *None*, *leader*, *e*)
else do {
let j = *the j*;
ASSERT (*j* ∈# 𝒱);
let nnxt = *hp-read-prev j arr*;
((𝒱, *arr*, *h*), *n*) ← *hp-link j leader* (𝒱, *arr*, *h*);
RETURN ((𝒱, *arr*, *h*), *nnxt*, *n*, *e*+*1*)
}
})
((𝒱, *arr*, *h*), *nxt*, *j*, *1*::*nat*);
RETURN (𝒱, *arr*, *Some leader*)
})›

lemma *vsids-pass*$_2$:
fixes *arr* :: ‹′*a*::*linorder multiset* × (′*a*, *nat*) *hp-fun* × ′*a option*›
assumes ‹*encoded-hp-prop-list2-conc arr* (𝒱′, *xs*)› **and** ‹*xs* ≠ []› **and** ‹*j* = *node* (*last xs*)›
shows ‹*vsids-pass*$_2$ *arr j* ≤ *SPEC*(λ(*arr*). *encoded-hp-prop-list-conc arr* (𝒱′, *ACIDS.pass*$_2$ *xs*))›
proof −
obtain *prevs nxts childs scores* 𝒱 **where**
arr: ‹*arr* = (𝒱, (*prevs*, *nxts*, *childs*, *scores*), *None*)› **and**
dist: ‹*distinct-mset* ($\sum_{\#}$ (*mset-nodes* ‘# (*mset* (*xs*))))› **and**
𝒱: ‹*set-mset* (*sum-list* (*map mset-nodes xs*)) ⊆ *set-mset* 𝒱› **and**
[*simp*]: ‹𝒱′ = 𝒱›
by (*cases arr*) (*use assms* **in** ‹*auto simp*: *ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def encoded-hp-prop-def*›)
have *prevs-lastxs*: ‹*prevs* (*node* (*last xs*)) = *map-option node* (*option-last* (*butlast xs*))›
using *assms*
by (*cases xs rule*: *rev-cases*; *cases* ‹*last xs*›)
(*auto simp*: *encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def arr*)

define *I* **where** ‹*I* ≡ (λ(*arr*, *nnxt*::′*a option*, *leader*, *e*′). *let e* = *length xs* − *e*′ *in*
encoded-hp-prop-list2-conc arr (𝒱, *take e xs* @ [*the* (*ACIDS.pass*$_2$ (*drop e xs*))]) ∧ *nnxt* = *map-option node* (*option-last* (*take e xs*)) ∧
leader = *node* (*the* (*ACIDS.pass*$_2$ (*drop e xs*))) ∧
e ≤ (*length xs*) ∧ (*nnxt* = *None* ⟷ *e* = *0*) ∧ *e*′ > *0*)›
have *I0*: ‹*I* ((𝒱, (*prevs*, *nxts*, *childs*, *scores*), *None*), *hp-read-prev j* (*prevs*, *nxts*, *childs*, *scores*), *j*, *1*)›
using *assms prevs-lastxs* **unfolding** *I-def prod.simps Let-def*
by (*auto simp*: *arr butlast-Nil-iff*)
have *j*𝒱: ‹*j* ∈# 𝒱›
using *assms* **by** (*cases xs rule*: *rev-cases*) (*auto simp*: *encoded-hp-prop-list2-conc-def arr*)
have *links-pre1*: ‹*encoded-hp-prop-list2-conc* (𝒱′, *arr*′, *h*′)
(𝒱, *take* (*length xs* − *Suc e*) *xs* @
xs ! (*length xs* − *Suc e*) #
the (*ACIDS.pass*$_2$ (*drop* (*length xs* − *e*) *xs*)) # [])› (**is** *?H1*) **and**
links-pre2: ‹*the x1b* = *node* (*xs* ! (*length xs* − *Suc e*))› (**is** *?H2*) **and**

links-pre3: ‹*leader* = *node* (*the* (*ACIDS.pass*$_2$ (*drop* (*length xs* − *e*) *xs*)))› (**is** *?H3*)
if
I: ‹*I s*› **and**
brk: ‹*case s of* (*x*, *xa*) ⇒ (*case x of* (𝒱, *arr*, *h*) ⇒ λ(*j*, *leader*, *e*). *j* ≠ *None*) *xa*› **and**
st: ‹*s* = (*a*, *b*)›
‹*x2b* = (*leader*, *e*)›
‹*b* = (*x1b*, *x2b*)›
‹*xy* = (*arr*′, *h*′)›
‹*a* = (𝒱′, *xy*)› **and**
no-None: ‹*x1b* ≠ *None*›
for *s a b* 𝒱′ *xy arr*′ *h*′ *x1b x2b x1c x2c e leader*
proof −
have ‹*e* < *length xs*› ‹*length xs* − *e* < *length xs*›
using *I brk no-None*
unfolding *st I-def*
by (*auto simp*: *I-def Let-def*)
then have *take-Suc*: ‹*take* (*length xs* − *e*) *xs* = *take* (*length xs* − *Suc e*) *xs* @ [*xs* ! (*length xs* − *Suc e*)]›
using *I brk take-Suc-conv-app-nth*[*of length xs* − *Suc e xs*]
unfolding *st*
apply (*cases* ‹*take* (*length xs* − *e*) *xs*› *rule*: *rev-cases*)
apply (*auto simp*: *I-def Let-def*)
done

then show *?H1*
using *I brk* **unfolding** *st*
apply (*cases* ‹*take* (*length xs* − *e*) *xs*› *rule*: *rev-cases*)
apply (*auto simp*: *I-def Let-def*)
done
show *?H2*
using *I brk* **unfolding** *st I-def Let-def*
by (*auto simp*: *take-Suc*)
show *?H3*
using *I brk* **unfolding** *st I-def Let-def*
by (*auto simp*: *take-Suc*)
qed
have *I-Suc*: ‹*I* ((*x1d*, *x1e*, *x2e*), *hp-read-prev* (*the x1b*) *x1a*, *new-leader*, *e* + *1*)›
if
I: ‹*I s*› **and**
brk: ‹*case s of* (*x*, *xa*) ⇒ (*case x of* (𝒱, *arr*, *h*) ⇒ λ(*j*, *leader*, *e*). *j* ≠ *None*) *xa*› **and**
st: ‹*s* = (*a*, *b*)›
‹*x2b* = (*x1c*, *e*)›
‹*b* = (*x1b*, *x2b*)›
‹*x2* = (*x1a*, *x2a*)›
‹*a* = (𝒱′, *x2*)›
‹*linkedn* = (*linked*, *new-leader*)›
‹*x2d* = (*x1e*, *x2e*)›
‹*linked* = (*x1d*, *x2d*)› **and**
no-None: ‹*x1b* ≠ *None*› **and**
‹*case linkedn of*
(*arr*, *n*) ⇒
encoded-hp-prop-list2-conc arr
(𝒱, *take* (*length xs* − *Suc e*) *xs* @
[*ACIDS.link* (*xs* ! (*length xs* − *Suc e*)) (*the* (*ACIDS.pass*$_2$ (*drop* (*length xs* − *e*) *xs*)))]) ∧
n =
node

(*ACIDS.link* (*xs* ! (*length xs* − *Suc e*)) (*the* (*ACIDS.pass*$_2$ (*drop* (*length xs* − *e*) *xs*))))›
for *s a b* $\mathcal{V}'$ *x2 x1a x2a x1b x2b x1c e linkedn linked new-leader x1d x2d x1e x2e*
proof −
have *e*: ‹*e* < *length xs*› ‹*length xs* − *e* < *length xs*›
using *I brk no-None*
unfolding *st I-def*
by (*auto simp*: *I-def Let-def*)
then have [*simp*]: ‹*ACIDS.link* (*xs* ! (*length xs* − *Suc e*)) (*the* (*ACIDS.pass*$_2$ (*drop* (*length xs* − *e*) *xs*))) =
the (*ACIDS.pass*$_2$ (*drop* (*length xs* − *Suc e*) *xs*))›
using *that*
by (*auto simp*: *I-def Let-def simp flip*: *Cons-nth-drop-Suc split*: *option.split*)
have [*simp*]: ‹*hp-read-prev* (*node* (*last* (*take* (*length xs* − *e*) *xs*))) *x1a* = *map-option node* (*option-last* (*take* (*length xs* − *Suc e*) *xs*))›
using *e I take-Suc-conv-app-nth*[*of length xs* − *Suc e xs*] **unfolding** *I-def st Let-def*
by (*cases* ‹(*take* (*length xs* − *e*) *xs*)› *rule*: *rev-cases*; *cases* ‹*last* (*take* (*length xs* − *e*) *xs*)›)
(*auto simp*: *encoded-hp-prop-list2-conc-def*
encoded-hp-prop-list-def)
show *?thesis*
using *that e* **by** (*auto simp*: *I-def Let-def*)
qed

show *?thesis*
unfolding *vsids-pass*$_2$*-def arr prod.simps*
apply (*refine-vcg WHILET-rule*[**where** *I*=*I* **and** *R* = ‹*measure* (λ(*arr*, *nnxt*::′*a option*, -, *e*). *length xs* −*e*)›]
hp-link)
subgoal using *j*$\mathcal{V}$ **by** *auto*
subgoal by *auto*
subgoal by (*rule I0*)
subgoal by *auto*
subgoal by *auto*
subgoal for *s a b x1 x2 x1a x2a x1b x2b x1c x2c*
by (*cases* ‹*take* (*length xs* − *x2c*) *xs*› *rule*: *rev-cases*)
(*auto simp*: *I-def Let-def encoded-hp-prop-list2-conc-def*)
apply (*rule links-pre1*; *assumption*)
subgoal
by (*rule links-pre2*)
subgoal
by (*rule links-pre3*)
subgoal
by (*rule I-Suc*)
subgoal for *s a b* $\mathcal{V}'$ *x2 x1a x2a x1b x2b x1c e linkedn linked new-leader x1d x2d x1e x2e*
by (*auto simp*: *I-def Let-def*)
subgoal using *assms ACIDS.mset-nodes-pass*$_2$[*of xs*] **by** (*auto simp*: *I-def Let-def*
encoded-hp-prop-list-conc-def encoded-hp-prop-list2-conc-def
split: *option.split simp del*: *ACIDS.mset-nodes-pass*$_2$)
done
qed

definition *merge-pairs* **where**
‹*merge-pairs arr j* = *do* {
(*arr*, *j*) ← *vsids-pass*$_1$ *arr j*;
vsids-pass$_2$ *arr j*
}›

lemma *vsids-merge-pairs*:
  fixes *arr* :: ‹$'a$::*linorder multiset* × ($'a$, *nat*) *hp-fun* × $'a$ *option*›
  assumes ‹*encoded-hp-prop-list2-conc arr* ($\mathcal{V}'$, *xs*)› **and** ‹$xs \neq []$› **and** ‹j = *node* (*hd xs*)›
  shows ‹*merge-pairs arr j* ≤ *SPEC*(λ(*arr*). *encoded-hp-prop-list-conc arr* ($\mathcal{V}'$, *ACIDS.merge-pairs xs*))›
proof −
  show *?thesis*
    unfolding *merge-pairs-def*
    apply (*refine-vcg vsids-pass*$_1$ *vsids-pass*$_2$[*of* - $\mathcal{V}'$ *ACIDS.pass*$_1$ *xs*])
    apply (*rule assms*)+
    subgoal by *auto*
    subgoal using *assms* **by** (*cases xs rule*: *ACIDS.pass*$_1$*.cases*) *auto*
    subgoal using *assms* **by** *auto*
    subgoal by (*auto simp*: *ACIDS.pass12-merge-pairs*)
    done
qed

definition *hp-update-child* **where**
  ‹*hp-update-child i nxt* = (λ(*prevs*, *nxts*, *childs*, *scores*). (*prevs*, *nxts*, *childs*(*i*:=*nxt*), *scores*))›

definition *vsids-pop-min* :: ‹-› **where**
  ‹*vsids-pop-min* = (λ($\mathcal{V}$::$'a$ *multiset*, *arr* :: ($'a$, $'b$::*order*) *hp-fun*, *h* :: $'a$ *option*). *do* {
  if h = None then RETURN (*None*, ($\mathcal{V}$, *arr*, *h*))
  else do {
      ASSERT (*the h* ∈# $\mathcal{V}$);
      let j = hp-read-child (*the h*) *arr*;
      if j = None then RETURN (*h*, ($\mathcal{V}$, *arr*, *None*))
      else do {
        ASSERT (*the j* ∈# $\mathcal{V}$);
        let arr = hp-update-prev (*the h*) *None arr*;
        let arr = hp-update-child (*the h*) *None arr*;
        let arr = hp-update-parents (*the j*) *None arr*;
        arr ← *merge-pairs* ($\mathcal{V}$, *arr*, *None*) (*the j*);
        RETURN (*h*, *arr*)
      }
    }
  })›

lemma *node-remove-key-itself-iff*[*simp*]: ‹*remove-key* (*y*) *z* ≠ *None* ⟹ *node z* = *node* (*the* (*remove-key* (*y*) *z*)) ⟷ *y* ≠ *node z*›
  by (*cases z*) *auto*

lemma *vsids-pop-min*:
  fixes *arr* :: ‹$'a$::*linorder multiset* × ($'a$, *nat*) *hp-fun* × $'a$ *option*›
  assumes ‹*encoded-hp-prop-list-conc arr* ($\mathcal{V}$, *h*)›
  shows ‹*vsids-pop-min arr* ≤ *SPEC*(λ(*j*, *arr*). *j* = (*if h = None then None else Some* (*get-min2 h*)) ∧ *encoded-hp-prop-list-conc arr* ($\mathcal{V}$, *ACIDS.del-min h*))›
proof −
  show *?thesis*
    unfolding *vsids-pop-min-def*
    apply (*refine-vcg vsids-merge-pairs*[*of* - $\mathcal{V}$ ‹*case the h of Hp* - - *child* ⇒ *child*›])
    subgoal using *assms* **by** (*cases h*) (*auto simp*: *encoded-hp-prop-list-conc-def*)
    subgoal using *assms* **by** (*auto simp*: *encoded-hp-prop-list-conc-def split*: *option.splits*)
    subgoal using *assms* **by** (*auto simp*: *encoded-hp-prop-list-conc-def split*: *option.splits*)
    subgoal using *assms* **by** (*auto simp*: *encoded-hp-prop-list-conc-def get-min2-alt-def split*: *option.splits*)

subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms encoded-hp-prop-list-remove-min*[*of* $\mathcal{V}$ ‹*node* (*the h*)› ‹*score* (*the h*)› ‹*hps* (*the h*)› ‹{#}›
‹*fst* (*fst* (*snd arr*))› ‹(*fst o snd*) (*fst* (*snd arr*))› ‹(*fst o snd o snd*) (*fst* (*snd arr*))›
‹(*fst o snd o snd o snd*) (*fst* (*snd arr*))›
‹(*snd o snd o snd o snd*) (*fst* (*snd arr*))›]
by (*cases* ‹*the h*›; *cases* ‹*fst* (*snd arr*)›)
(*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list2-conc-def hp-update-parents-def hp-update-nxt-def hp-update-score-def hp-update-child-def hp-update-prev-def get-min2-alt-def split*: *option.splits if-splits*)
subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms* **by** (*cases* ‹*h*›; *cases* ‹*the h*›)
(*auto simp*: *get-min2-alt-def ACIDS.pass12-merge-pairs encoded-hp-prop-list-conc-def split*: *option.splits*)
done
qed

Unconditionnal version of the previous function

definition *vsids-pop-min2* :: ‹-› **where**
‹*vsids-pop-min2* = (λ($\mathcal{V}$::'*a multiset*, *arr* :: ('*a*, '*b*::*order*) *hp-fun*, *h* :: '*a option*). *do* {
ASSERT (*h*≠*None*);
ASSERT (*the h* ∈# $\mathcal{V}$);
let j = *hp-read-child* (*the h*) *arr*;
if j = *None then RETURN* (*the h*, ($\mathcal{V}$, *arr*, *None*))
else do {
ASSERT (*the j* ∈# $\mathcal{V}$);
let arr = *hp-update-prev* (*the h*) *None arr*;
let arr = *hp-update-child* (*the h*) *None arr*;
let arr = *hp-update-parents* (*the j*) *None arr*;
arr ← *merge-pairs* ($\mathcal{V}$, *arr*, *None*) (*the j*);
RETURN (*the h*, *arr*)
}
}
)›

lemma *vsids-pop-min2*:
fixes *arr* :: ‹'*a*::*linorder multiset* × ('*a*, *nat*) *hp-fun* × '*a option*›
assumes ‹*encoded-hp-prop-list-conc arr* ($\mathcal{V}$, *h*)› **and** ‹*h* ≠ *None*›
shows ‹*vsids-pop-min2 arr* ≤ *SPEC*(λ(*j*, *arr*). *j* = (*get-min2 h*) ∧ *encoded-hp-prop-list-conc arr* ($\mathcal{V}$, *ACIDS.del-min h*))›
proof −
show *?thesis*
unfolding *vsids-pop-min2-def*
apply (*refine-vcg vsids-merge-pairs*[*of* - $\mathcal{V}$ ‹*case the h of Hp - - child* ⇒ *child*›])
subgoal using *assms* **by** (*cases h*) (*auto simp*: *encoded-hp-prop-list-conc-def*)
subgoal using *assms* **by** (*auto simp*: *encoded-hp-prop-list-conc-def split*: *option.splits*)
subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)

subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms encoded-hp-prop-list-remove-min*[*of* $\mathcal{V}$ ‹*node* (*the h*)› ‹*score* (*the h*)› ‹*hps* (*the h*)› ‹{#}›
‹*fst* (*fst* (*snd arr*))› ‹(*fst o snd*) (*fst* (*snd arr*))› ‹(*fst o snd o snd*) (*fst* (*snd arr*))›
‹(*fst o snd o snd o snd*) (*fst* (*snd arr*))›
‹(*snd o snd o snd o snd*) (*fst* (*snd arr*))›]
by (*cases* ‹*the h*›; *cases* ‹*fst* (*snd arr*)›)
(*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list2-conc-def hp-update-parents-def hp-update-nxt-def hp-update-score-def hp-update-child-def hp-update-prev-def get-min2-alt-def split*: *option.splits if-splits*)
subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms* **by** (*cases* ‹*the h*›) (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def get-min2-alt-def split*: *option.splits*)
subgoal using *assms* **by** (*cases* ‹*h*›; *cases* ‹*the h*›)
(*auto simp*: *get-min2-alt-def ACIDS.pass12-merge-pairs encoded-hp-prop-list-conc-def split*: *option.splits*)
done
qed

lemma *in-remove-key-in-find-keyD*:
‹$m' \in\#$ (*if remove-key a h* = *None then* {#} *else* {#*the* (*remove-key a h*)#}) +
(*if find-key a h* = *None then* {#} *else* {#*the* (*find-key a h*)#}) $\Longrightarrow$
distinct-mset (*mset-nodes h*) $\Longrightarrow$
$x' \in\#$ *mset-nodes* $m' \Longrightarrow x' \in\#$ *mset-nodes h*›
using *find-remove-mset-nodes-full*[*of h a* ‹*the* (*remove-key a h*)› ‹*the* (*find-key a h*)›]
in-remove-key-in-nodes[*of a h x'*]
apply (*auto split*: *if-splits simp*: *find-key-None-remove-key-ident*)
apply (*metis hp-node-None-notin2 hp-node-in-find-key0*)
by (*metis union-iff*)

lemma *map-option-node-map-option-node-iff*:
‹($x \neq$ *None* $\Longrightarrow$ *distinct-mset* (*mset-nodes* (*the x*))) $\Longrightarrow$ ($x \neq$ *None* $\Longrightarrow$ $y \neq$ *node* (*the x*)) $\Longrightarrow$
map-option node x = *map-option node* (*map-option* (λx. *the* (*remove-key y x*)) *x*)›
by (*cases x*; *cases* ‹*the x*›) *auto*

lemma *distinct-mset-hp-parent*: ‹*distinct-mset* (*mset-nodes h*) $\Longrightarrow$ *hp-parent a h* = *Some ya* $\Longrightarrow$ *distinct-mset* (*mset-nodes ya*)›
apply (*induction a h arbitrary*: *ya rule*: *hp-parent.induct*)
apply (*auto simp*: *hp-parent-simps-if hp-parent-children-cons split*: *if-splits option.splits*)
apply (*metis* (*no-types*, *lifting*) *WB-List-More.distinct-mset-union2 distinct-mset-union hp-parent-children-Some-iff in-list-in-setD list.map*(*2*) *map-append sum-list.Cons sum-list-append*)
by (*metis distinct-mset-union*)

lemma *in-find-key-children-same-hp-parent*:
‹*hp-parent k* (*Hp x n c*) = *None* $\Longrightarrow$
$x' \in\#$ *mset-nodes* $m' \Longrightarrow$
$x \notin\#$ *sum-list* (*map mset-nodes c*) $\Longrightarrow$
distinct-mset (*sum-list* (*map mset-nodes c*)) $\Longrightarrow$
find-key-children k c = *Some* $m' \Longrightarrow$ *hp-parent* x' (*Hp x n c*) = *hp-parent* x' m'›
apply (*induction k c rule*: *find-key-children.induct*)

subgoal
by (*auto split*: *if-splits option.splits simp*: *hp-parent-simps-single-if hp-parent-children-cons*)
subgoal for *k xa na c xs*
apply (*auto split*: *if-splits option.splits simp*: *hp-parent-simps-single-if hp-parent-children-cons*)
apply (*metis mset-nodes-find-key-children-subset mset-subset-eqD option.sel option.simps*(*3*) *sum-image-mset-sum-ma*
apply (*metis* (*no-types*, *lifting*) *ACIDS.hp-node-find-key-children find-key-children.simps*(*1*) *find-key-children-None-or-*
hp.sel(*1*) *hp-node-None-notin2 hp-node-children-simps*(*3*) *hp-node-node-itself hp-parent-children-in-first-child*
hp-parent-in-nodes list.exhaust-sel
list.simps(*9*) *mset-nodes-find-key-children-subset mset-subset-eqD node-in-mset-nodes option.sel*
sum-image-mset-sum-map sum-list-simps(*2*))
apply (*metis hp-node-None-notin2 hp-node-children-None-notin2 hp-node-in-find-key-children sum-image-mset-sum-m*
apply (*smt* (*verit*, *ccfv-threshold*) *basic-trans-rules*(*31*) *find-key-children.elims find-key-children.simps*(*2*)
hp.exhaust-sel hp.sel(*1*)
hp-parent-children-in-first-child hp-parent-in-nodes list.distinct(*1*) *list.exhaust-sel list.simps*(*9*)
mset-nodes-find-key-children-subset
option.sel option.simps(*2*) *set-mset-mono sum-image-mset-sum-map sum-list-simps*(*2*))
apply (*metis disjunct-not-in distinct-mset-add find-key-noneD find-key-none-iff mset-map mset-nodes-find-key-children*
mset-subset-eqD node-hd-in-sum option.sel sum-mset-sum-list)
apply (*smt* (*verit*, *ccfv-threshold*) *basic-trans-rules*(*31*) *find-key-children.elims find-key-children.simps*(*2*)
hp.exhaust-sel hp.sel(*1*)
hp-parent-children-in-first-child hp-parent-in-nodes list.distinct(*1*) *list.exhaust-sel list.simps*(*9*)
mset-nodes-find-key-children-subset
option.sel option.simps(*2*) *set-mset-mono sum-image-mset-sum-map sum-list-simps*(*2*))
apply (*smt* (*verit*) *ACIDS.hp-node-find-key-children distinct-mset-add ex-hp-node-children-Some-in-mset-nodes*
find-key-children.simps(*1*) *find-key-children-None-or-itself*
find-key-none-iff hp.sel(*1*) *hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-simps*(*3*)
hp-node-in-find-key-children hp-node-node-itself
hp-parent-children-in-first-child hp-parent-in-nodes list.exhaust-sel list.simps(*9*) *option.sel option-last-Nil*
option-last-Some-iff(*2*) *sum-list-simps*(*2*))
apply (*metis Duplicate-Free-Multiset.distinct-mset-union2 hp-parent-children-hd-None option.simps*(*2*)
sum-image-mset-sum-map union-commute)
apply (*metis disjunct-not-in distinct-mset-add hp-parent-children-None-notin if-Some-None-eq-None*
mset-map mset-nodes-find-key-children-subset mset-subset-eqD
option.sel sum-mset-sum-list)
apply (*metis Duplicate-Free-Multiset.distinct-mset-union2 hp.sel*(*1*) *hp-parent-in-nodes mset-nodes-find-key-children-s*
mset-subset-eqD option.sel option.simps(*2*) *sum-image-mset-sum-map union-commute*)
apply (*metis basic-trans-rules*(*31*) *mset-nodes-find-key-children-subset option.distinct*(*1*) *option.sel*
set-mset-mono sum-image-mset-sum-map)
apply (*metis distinct-mset-add find-key-noneD find-key-none-iff hp-parent-children-None-notin hp-parent-children-skip-*
hp-parent-children-skip-last
mset-map mset-nodes-find-key-children-subset mset-subset-eqD option.sel sum-mset-sum-list)
apply (*simp add*: *distinct-mset-add*)
using *distinct-mset-union* **by** *blast*
done

lemma *in-find-key-same-hp-parent*:
‹x' ∈# *mset-nodes* m' ⟹
distinct-mset (*mset-nodes h*) ⟹
find-key a h = *Some* m' ⟹
hp-parent a h = *None* ⟹
∃ *y*. *hp-prev a h* = *Some y* ⟹
hp-parent x' *h* = *hp-parent* x' m'›
by (*induction a h rule*: *find-key.induct*)
(*auto split*: *if-splits intro*: *in-find-key-children-same-hp-parent*)

lemma *in-find-key-children-same-hp-parent2*:
‹x′ ≠ k ⟹
x′ ∈# mset-nodes m′ ⟹
x ∉# sum-list (map mset-nodes c) ⟹
distinct-mset (sum-list (map mset-nodes c)) ⟹
find-key-children k c = Some m′ ⟹ hp-parent x′ (Hp x n c) = hp-parent x′ m′›
apply (*induction k c rule*: *find-key-children.induct*)
subgoal
by (*auto split*: *if-splits option.splits simp*: *hp-parent-simps-single-if hp-parent-children-cons*)
subgoal for *k xa na c xs*
apply (*auto split*: *if-splits option.splits simp*: *hp-parent-simps-single-if hp-parent-children-cons*)
apply (*metis add-diff-cancel-left′ distinct-mem-diff-mset hp-parent-children-None-notin*)
apply (*metis hp-node-None-notin2 hp-node-children-None-notin2 hp-node-in-find-key-children sum-image-mset-sum-m*
apply (*metis hp.sel*(*1*) *hp-parent-in-nodes2 mset-nodes-find-key-children-subset mset-subset-eqD option.sel option.simps*(*2*) *sum-image-mset-sum-map*)
apply (*metis disjunct-not-in distinct-mset-add find-key-noneD find-key-none-iff mset-map mset-nodes-find-key-children mset-subset-eqD node-hd-in-sum option.sel sum-mset-sum-list*)
apply (*metis hp-node-None-notin2 hp-node-children-None-notin2 hp-node-in-find-key-children sum-image-mset-sum-m*
apply (*metis hp.sel*(*1*) *hp-parent-in-nodes2 mset-nodes-find-key-children-subset mset-subset-eqD option.sel option.simps*(*2*) *sum-image-mset-sum-map*)
apply (*metis mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff*(*2*) *sum-image-mset-sum-map*)
apply (*metis basic-trans-rules*(*31*) *distinct-mset-union ex-hp-node-children-Some-in-mset-nodes hp.sel*(*1*) *hp-node-children-simps*(*1*) *hp-parent-in-nodes mset-nodes-find-key-children-subset option.sel option.simps*(*2*) *set-mset-mono sum-image-mset-sum-map union-commute*)
apply (*metis distinct-mset-union hp-parent-children-hd-None option-last-Nil option-last-Some-iff*(*2*) *sum-image-mset-sum-map*)
apply (*metis disjunct-not-in distinct-mset-add hp-parent-children-None-notin mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff*(*2*) *sum-image-mset-sum-map*)
apply (*metis distinct-mset-union hp.sel*(*1*) *hp-parent-in-nodes mset-nodes-find-key-children-subset mset-subset-eqD option.sel option.simps*(*2*) *sum-image-mset-sum-map*)
apply (*metis mset-nodes-find-key-children-subset mset-subset-eqD option.sel option-last-Nil option-last-Some-iff*(*2*) *sum-image-mset-sum-map*)
apply (*metis disjunct-not-in distinct-mset-add hp-node-None-notin2 hp-node-children-None-notin2 hp-node-in-find-key-children hp-parent-children-None-notin sum-image-mset-sum-map*)
apply (*metis distinct-mset-union hp-parent-children-hd-None option.simps*(*2*) *sum-image-mset-sum-map*)
using *distinct-mset-union* **by** *blast*
done

lemma *in-find-key-same-hp-parent2*:
‹x′ ∈# mset-nodes m′ ⟹
distinct-mset (mset-nodes h) ⟹
find-key a h = Some m′ ⟹
x′ ≠ a ⟹
hp-parent x′ h = hp-parent x′ m′›
by (*induction a h rule*: *find-key.induct*)
(*auto split*: *if-splits intro*: *in-find-key-children-same-hp-parent2*)

lemma *encoded-hp-prop-list-remove-find*:
fixes *h* :: ‹(′a, nat) hp› **and** *a arr* **and** *hs* :: ‹(′a, nat) hp multiset›
defines ‹arr_1 ≡ (if hp-parent a h = None then arr else hp-update-child (node (the (hp-parent a h))) (map-option node (hp-next a h)) arr)›
defines ‹arr_2 ≡ (if hp-prev a h = None then arr_1 else hp-update-nxt (node (the (hp-prev a h))) (map-option node (hp-next a h)) arr_1)›
defines ‹arr_3 ≡ (if hp-next a h = None then arr_2 else hp-update-prev (node (the (hp-next a h))) (map-option node (hp-prev a h)) arr_2)›

defines ‹arr_4 ≡ (*if hp-next a h = None then* arr_3 *else hp-update-parents* (*node* (*the* (*hp-next a h*))) (*map-option node* (*hp-parent a h*)) arr_3)›
defines ‹*arr′* ≡ *hp-update-parents a None* (*hp-update-prev a None* (*hp-update-nxt a None* arr_4))›
assumes *enc*: ‹*encoded-hp-prop-list* 𝒱 (*add-mset h* {#}) [] *arr*›
shows ‹*encoded-hp-prop-list* 𝒱 ((*if remove-key a h = None then* {#} *else* {#*the* (*remove-key a h*)#}) +
(*if find-key a h = None then* {#} *else* {#*the* (*find-key a h*)#})) []
arr′›
proof −
obtain *prevs nxts childs parents scores* **where**
arr: ‹*arr* = ((*prevs*, *nxts*, *childs*, *parents*, *scores*))› **and**
dist: ‹*distinct-mset* (*mset-nodes h*)› **and**
𝒱: ‹*set-mset* (*mset-nodes h*) ⊆ *set-mset* 𝒱›
by (*cases arr*) (*use assms* **in** ‹*auto simp*: *ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def encoded-hp-prop-def*›)
have *find-key-in-nodes*: ‹*find-key a h* ≠ *None* ⟹ *node* (*the* (*find-key a h*)) ∈# *mset-nodes h*›
by (*cases* ‹*a* ∈# *mset-nodes h*›)
(*use find-key-None-or-itself*[*of a h*] **in** ‹*auto simp del*: *find-key-None-or-itself*›)
have *in-find-key-in-nodes1*: ‹*x* ∈# *mset-nodes y* ⟹ *find-key a h = Some y* ⟹ *x* ∈# *mset-nodes h*› **for** *x y*
using *mset-nodes-find-key-subset*[*of a h*]
by *auto*
have [*simp*]: ‹*find-key a h = None* ⟹ *remove-key a h = Some h*›
by (*metis find-key.simps find-key-none-iff hp.exhaust-sel hp-node-None-notin2 hp-node-children-None-notin2 hp-node-children-simps2 option-last-Nil option-last-Some-iff* (*2*) *remove-key-notin-unchanged*)
have *HX1*: ‹
(*find-key* (*node m′*) *h* ≠ *None* ⟹
distinct-mset
(*mset-nodes* (*the* (*find-key* (*node m′*) *h*)) +
(*if hp-next* (*node m′*) *h = None then* {#}
else mset-nodes (*the* (*hp-next* (*node m′*) *h*))))) ⟹
x′ ∈# *mset-nodes m′* ⟹
x′ ∉# *mset-nodes y′* ⟹
find-key (*node y*) *h = Some y* ⟹
m′ = y′ ∨ *m′ = y* ⟹
hp-next (*node y*) *h* ≠ *None* ⟹
x′ = node (*the* (*hp-next* (*node y*) *h*)) ⟹
map-option node (*hp-prev* (*node y*) *h*) = *map-option node* (*hp-prev* (*node* (*the* (*hp-next* (*node y*) *h*))) *m′*)›
for *y y′ m′ x′*
by (*smt* (*z3*) *distinct-mset-iff mset-add node-in-mset-nodes option.distinct*(*1*) *option.sel union-mset-add-mset-left union-mset-add-mset-right*)
have
dist: ‹*distinct-mset* (*mset-nodes h*)› **and**
nxts: ‹(∀ *m′*∈#{#*h*#}. ∀ *x*∈#*mset-nodes m′*. *nxts x = map-option node* (*hp-next x m′*))› **and**
prevs: ‹(∀ *m*∈#{#*h*#}. ∀ *x*∈#*mset-nodes m*. *prevs x = map-option node* (*hp-prev x m*))› **and**
childs: ‹(∀ *m*∈#{#*h*#}. ∀ *x*∈#*mset-nodes m*. *childs x = map-option node* (*hp-child x m*))› **and**
parents: ‹(∀ *m*∈#{#*h*#}. ∀ *x*∈#*mset-nodes m*. *parents x = map-option node* (*hp-parent x m*))› **and**
scores: ‹(∀ *m*∈#{#*h*#}. ∀ *x*∈#*mset-nodes m*. *scores x = hp-score x m*)› **and**
empty-outside: ‹*empty-outside* ($\sum_{\#}$ (*mset-nodes* ‘# {#*h*#} + *mset-nodes* ‘# *mset* [])) *prevs*›
‹*empty-outside* ($\sum_{\#}$ (*mset-nodes* ‘# {#*h*#} + *mset-nodes* ‘# *mset* [])) *parents*›
using *enc* **unfolding** *encoded-hp-prop-list-def prod.simps arr* **by** *auto*
let *?a* = ‹(*if remove-key a h = None then* {#} *else* {#*the* (*remove-key a h*)#}) +
(*if find-key a h = None then* {#} *else* {#*the* (*find-key a h*)#})›
have *H*: ‹*remove-key a h* ≠ *None* ⟹ *node* (*the* (*remove-key a h*)) ∈# *mset-nodes h*›

by (*metis remove-key.simps get-min2.simps hp.exhaust-sel option.collapse option.distinct*(*2*) *remove-key-notin-unchanged*)
show *?thesis*
supply [[*goals-limit=1*]]
using *dist*
unfolding *arr hp-update-child-def hp-update-nxt-def hp-update-prev-def case-prod-beta hp-update-parents-def encoded-hp-prop-list-def prod.simps* **apply** −

proof (*intro conjI impI ballI*)
show ‹*distinct-mset* ($\sum_\#$ (*mset-nodes* '# *?a* +
mset-nodes '# *mset* []))›
using *dist*
apply (*auto simp*: *find-remove-mset-nodes-full*)
apply (*metis distinct-mset-mono′ mset-nodes-find-key-subset option.distinct*(*2*) *option.sel*)
done
next
show ‹*set-mset* ($\sum_\#$ (*mset-nodes* '#
((*if remove-key a h* = *None then* {#} *else* {#*the* (*remove-key a h*)#}) +
(*if find-key a h* = *None then* {#} *else* {#*the* (*find-key a h*)#})) +
mset-nodes '# *mset* []))
⊆ *set-mset* $\mathcal{V}$›
using $\mathcal{V}$ **apply** (*auto dest*: *in-find-key-in-nodes1*)
apply (*metis Set.basic-monos*(*7*) *in-remove-key-in-nodes option.distinct*(*2*) *option.sel*)
done
next
fix *m′* **and** *x′*
assume ‹*m′*∈#*?a*› **and** ‹*x′* ∈# *mset-nodes m′*›
then show ‹*fst* (*snd arr′*) *x′* = *map-option node* (*hp-next x′ m′*)›
using *nxts dist H*
hp-next-find-key[*of h a x′*] *hp-next-find-key-itself*[*of h a*]
in-remove-key-in-nodes[*of a h x′*] *in-find-key-notin-remove-key*[*of h a x′*]
in-find-key-in-nodes[*of a h x′*]
unfolding *assms*(*1*−*5*) *arr*
using *hp-next-remove-key-other*[*of h a x′*] *find-key-None-or-itself*[*of a h*]
hp-next-find-key-itself[*of h a*] *has-prev-still-in-remove-key*[*of h a*]
in-remove-key-changed[*of a h*]
hp-parent-itself[*of h*] *remove-key-None-iff*[*of a h*] *find-key-head-node-iff*[*of h m′*]
by (*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def hp-update-parents-def map-option.compositionality comp-def map-option-node-hp-next-remove-key*
split: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
next
fix *m′* **and** *x′*
assume *M′*: ‹*m′*∈#*?a*› ‹*x′* ∈# *mset-nodes m′*›
then show ‹*fst arr′ x′* = *map-option node* (*hp-prev x′ m′*)›
using *prevs H dist*
hp-prev-find-key[*of h a x′*]
in-remove-key-in-nodes[*of a h x′*] *in-find-key-notin-remove-key*[*of h a x′*]
in-find-key-in-nodes[*of a h x′*]
unfolding *assms*(*1*−*5*) *arr*
using *hp-prev-remove-key-other*[*of h a x′*] *find-key-None-or-itself*[*of a h*]
hp-prev-find-key-itself[*of h a*] *has-prev-still-in-remove-key*[*of h a*]
in-remove-key-changed[*of a h*]
hp-parent-itself[*of h*] *remove-key-None-iff*[*of a h*]
find-key-head-node-iff[*of h m′*]
using *hp-prev-and-next-same-node*[*of h x′ m′* ‹*the* (*hp-next* (*node m′*) *h*)›]
distinct-mset-find-node-next[*of h* ‹*node m′*› ‹*the* (*find-key* (*node m′*) *h*)›]

apply (*simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def hp-update-parents-def map-option.compositionality comp-def map-option-node-hp-prev-remove-key split*: *if-splits del*: *find-key-None-or-itself hp-parent-itself*)
apply (*intro conjI impI allI*)
subgoal
by (*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-prev-remove-key split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
subgoal
by (*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-prev-remove-key split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
apply (*intro conjI impI allI*)
subgoal
by (*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-prev-remove-key split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
subgoal
unfolding *eq-commute*[*of - x'*]
by (*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-prev-remove-key split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
subgoal
using *node-in-mset-nodes*[*of* ‹*the* (*hp-next* (*node m'*) *h*)›]
unfolding *eq-commute*[*of - x'*]
by *auto*
subgoal
using *node-in-mset-nodes*[*of* ‹*the* (*hp-next* (*node m'*) *h*)›]
unfolding *eq-commute*[*of - x'*]
by *auto*
subgoal for *y y'*
apply (*clarsimp simp add*: *atomize-not hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-prev-remove-key hp-update-parents-def split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
apply (*intro conjI impI*)
using *HX1*[*of y x' y' m'*]
apply (*auto simp add*: *atomize-not hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-prev-remove-key hp-update-parents-def split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
done
subgoal
by (*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-prev-remove-key split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
subgoal
by (*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-prev-remove-key split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
done
next
fix *m'* **and** *x'*
assume *M'*: ‹*m'*∈#*?a*› ‹*x'* ∈# *mset-nodes m'*›
have *helper1*: ‹*hp-parent* (*node yb*) *yyy* = *None*›
if
‹*distinct-mset* (*mset-nodes yyy*)› **and**
‹*node y* ∈# *mset-nodes h*› **and**

‹hp-parent (node yyy) h = Some y› **and**
‹hp-child (node y) h = Some yb›
for y :: ‹('a, nat) hp› **and** ya :: ‹('a, nat) hp› **and** yb :: ‹('a, nat) hp› **and** z :: ‹('a, nat) hp› **and** yyy
using childs[simplified]
by (metis dist hp-child-hp-parent hp-parent-itself option.map-sel option.sel option-last-Nil option-last-Some-iff(1)
that)
have helper2: ‹hp-child (node ya) yyy ≠ hp-child (node ya) h›
if
‹distinct-mset (mset-nodes yyy)› **and**
‹hp-parent (node yyy) h = Some ya›
‹node ya ∈# mset-nodes h›
for y :: ‹('a, nat) hp› **and** ya :: ‹('a, nat) hp› **and** yyy yya
using childs[simplified]
by (metis dist that hp-child-hp-parent hp-parent-hp-child hp-parent-itself map-option-is-None option.map-sel option.sel option-last-Nil option-last-Some-iff(1))
have helper4: ‹map-option node (map-option (λx. the (remove-key (node yy) x)) (hp-child (x') h)) = map-option node (hp-child (x') h)›
if
‹∃ y. hp-child (x') h = Some y ⟹ ∃ z. hp-parent (node (the (hp-child (x') h))) h = Some z ∧ node z = x'› **and**
‹node h = node yya ⟹ find-key (node yya) h ≠ Some yya› **and**
‹hp-parent (node yy) h = None›
for yya yy x'
using that childs[simplified] dist **apply** −
using distinct-sum-next-prev-child[of h x']
apply (auto simp: map-option-node-remove-key-iff)
apply (subst eq-commute)
apply (rule ccontr)
apply (subst (asm) map-option-node-remove-key-iff)
apply simp
apply (meson distinct-mset-add)
by (auto simp: remove-key-None-iff)

have ‹find-key a h ≠ None ⟹ distinct-mset (mset-nodes (the (find-key a h)))›
by (meson dist distinct-mset-mono' mset-nodes-find-key-subset)

then show ‹fst (snd (snd arr')) x' = map-option node (hp-child x' m')›
using childs dist H M'
hp-child-find-key[of h a x']
in-remove-key-in-nodes[of a h x'] in-find-key-notin-remove-key[of h a x']
in-find-key-in-nodes[of a h x']
hp-parent-hp-child[of h x'] hp-child-hp-parent[of h x']
hp-child-hp-parent[of h x']
hp-parent-hp-child[of ‹the (find-key a h)› x']
unfolding assms(1−5) arr
using hp-child-remove-key-other[of h a x'] find-key-None-or-itself[of a h]
hp-next-find-key-itself[of h a] has-prev-still-in-remove-key[of h a]
in-remove-key-changed[of a h]
hp-parent-itself[of h] remove-key-None-iff[of a h] find-key-head-node-iff[of h m']

apply (simp split: if-splits(2) del: find-key-None-or-itself hp-parent-itself)
apply (clarsimp simp add: hp-update-child-def hp-update-prev-def hp-update-nxt-def hp-update-parents-def
map-option.compositionality comp-def map-option-node-hp-next-remove-key
split: if-splits simp del: find-key-None-or-itself hp-parent-itself)

apply (*clarsimp simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def hp-update-parents-def map-option.compositionality comp-def map-option-node-hp-next-remove-key split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
apply (*solves* ‹*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def helper2 map-option.compositionality comp-def map-option-node-hp-next-remove-key hp-update-parents-def split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*›)[]
apply (*solves* ‹*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def helper2 map-option.compositionality comp-def map-option-node-hp-next-remove-key hp-update-parents-def split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*›)[]

apply (*clarsimp simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-next-remove-key hp-update-parents-def split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
apply (*intro conjI impI*)

subgoal for *yy yya*
apply *auto*
apply (*subst* (*asm*) *helper4*)
apply *assumption+*
apply *simp*
done
apply (*clarsimp simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-next-remove-key hp-update-parents-def split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
subgoal
using *distinct-sum-next-prev-child*[*of h x'*]
apply (*auto simp*: *remove-key-None-iff map-option-node-remove-key-iff*)
apply (*subst* (*asm*) *map-option-node-remove-key-iff*)
apply *simp*
apply (*meson distinct-mset-add*)
by (*auto simp*: *remove-key-None-iff*)
apply (*clarsimp simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-next-remove-key hp-update-parents-def split*: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
using *distinct-sum-next-prev-child*[*of h x'*]
apply (*auto simp*: *remove-key-None-iff map-option-node-remove-key-iff*)
apply (*subst* (*asm*) *map-option-node-remove-key-iff*)
apply *simp*
apply (*meson distinct-mset-add*)
by (*auto simp*: *remove-key-None-iff*)
subgoal by *auto*
subgoal for *y y'*
using *hp-child-remove-key-other*[*of h a x'*, *symmetric*]
apply (*auto simp*: *map-option.compositionality comp-def*)
apply (*subst* (*asm*) *map-option-node-map-option-node-iff*)
apply *auto*[]
apply (*smt* (*verit*, *del-insts*) *None-eq-map-option-iff node-remove-key-itself-iff option.distinct*(*2*) *option.exhaust-sel option.map-sel remove-key-None-iff*)
apply (*smt* (*verit*) *None-eq-map-option-iff node-remove-key-itself-iff option.exhaust-sel op-*

```
tion.simps(9) remove-key-None-iff)
        by (metis (no-types, lifting) map-option-cong node-remove-key-itself-iff option.sel option.simps(3)
remove-key-None-iff)
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal
        apply auto
        by (metis no-relative-ancestor-or-notin)
      subgoal
        apply auto
       by (smt (verit, del-insts) None-eq-map-option-iff hp.exhaust-sel hp-child-remove-is-remove-hp-child
node-remove-key-itself-iff option.exhaust-sel option.map(2) option.simps(1))
      subgoal
          by (smt (verit, ccfv-SIG) None-eq-map-option-iff node-remove-key-itself-iff option.exhaust-sel
option.map-sel remove-key-None-iff)
      subgoal
          by (smt (verit, ccfv-SIG) None-eq-map-option-iff node-remove-key-itself-iff option.exhaust-sel
option.map-sel remove-key-None-iff)
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
      subgoal by auto
    apply (clarsimp simp add: hp-update-child-def hp-update-prev-def hp-update-nxt-def hp-update-parents-def
          map-option.compositionality comp-def map-option-node-hp-next-remove-key
        split: if-splits simp del: find-key-None-or-itself hp-parent-itself)
      subgoal
        using distinct-sum-next-prev-child[of h x']
        apply (auto simp: remove-key-None-iff map-option-node-remove-key-iff)
        apply (subst (asm) map-option-node-remove-key-iff)
        apply simp
        apply (meson distinct-mset-add)
        by (auto simp: remove-key-None-iff)
    apply (clarsimp simp add: hp-update-child-def hp-update-prev-def hp-update-nxt-def hp-update-parents-def
          map-option.compositionality comp-def map-option-node-hp-next-remove-key
        split: if-splits simp del: find-key-None-or-itself hp-parent-itself)
      subgoal
        using distinct-sum-next-prev-child[of h x']
        apply (auto simp: remove-key-None-iff map-option-node-remove-key-iff)
        apply (subst (asm) map-option-node-remove-key-iff)
        apply simp
        apply (meson distinct-mset-add)
        by (auto simp: remove-key-None-iff)
    apply (clarsimp simp add: hp-update-child-def hp-update-prev-def hp-update-nxt-def hp-update-parents-def
          map-option.compositionality comp-def map-option-node-hp-next-remove-key
        split: if-splits simp del: find-key-None-or-itself hp-parent-itself)
      subgoal
        using distinct-sum-next-prev-child[of h x']
        apply (auto simp: remove-key-None-iff map-option-node-remove-key-iff)
        apply (subst (asm) map-option-node-remove-key-iff)
        apply simp
        apply (meson distinct-mset-add)
        by (auto simp: remove-key-None-iff)
    apply (clarsimp simp add: hp-update-child-def hp-update-prev-def hp-update-nxt-def hp-update-parents-def
```

map-option.compositionality comp-def map-option-node-hp-next-remove-key
split: *if-splits* *simp del*: *find-key-None-or-itself hp-parent-itself*)
subgoal
using *distinct-sum-next-prev-child*[*of h x'*]
apply (*auto simp*: *remove-key-None-iff map-option-node-remove-key-iff*)
apply (*subst* (*asm*) *map-option-node-remove-key-iff*)
apply *simp*
apply (*meson distinct-mset-add*)
by (*auto simp*: *remove-key-None-iff*)
done

have *helper1*: *False*
if
‹*distinct-mset* (*mset-nodes h*)› **and**
‹*node y* ∈# *mset-nodes m'*› **and**
‹*node y* ∈# *mset-nodes ya*› **and**
‹*remove-key a h* = *Some m'*› **and**
‹*find-key a h* = *Some ya*› **and**
‹*x'* = *node y*›
for *ya* :: ‹('*a*, *nat*) *hp*› **and** *y* :: ‹('*a*, *nat*) *hp*› **and** *yb* :: ‹('*a*, *nat*) *hp*›
by (*metis that Some-to-the in-find-key-notin-remove-key option-last-Nil option-last-Some-iff*(*2*))
have *helper3*: ‹*False*›
if
‹*distinct-mset* (*mset-nodes h*)› **and**
‹*x'* ∈# *mset-nodes m'*› **and**
‹*x'* ∈# *mset-nodes ya*› **and**
‹*remove-key a h* = *Some m'*› **and**
‹*find-key a h* = *Some ya*›
for *ya* :: ‹('*a*, *nat*) *hp*›
by (*metis that Some-to-the in-find-key-notin-remove-key option-last-Nil option-last-Some-iff*(*1*))
have *helperb4*: ‹*False*›
if
‹*h* = *m'*› **and**
‹*hp-next a m'* = *Some z*› **and**
‹*find-key a m'* = *None*›
for *z* :: ‹('*a*, *nat*) *hp*› **and** *y* :: ‹('*a*, *nat*) *hp*›
by (*metis that find-key-None-remove-key-ident hp-next-None-notin in-remove-key-changed option.sel option.simps*(*2*))
have [*simp*]: ‹*map-option* (λ*x*. *node* (*the* (*remove-key a x*))) (*hp-parent a h*) = *map-option node* (*hp-parent a h*)›
for *z* :: ‹('*a*, *nat*) *hp*›
by (*smt* (*verit*, *ccfv-SIG*) *None-eq-map-option-iff distinct-mset-find-node-next distinct-mset-union find-key-None-or-itself*
find-key-None-remove-key-ident find-key-notin hp-child-find-key hp-child-hp-parent hp-parent-hp-child hp-parent-in-nodes
hp-parent-itself in-remove-key-changed node-remove-key-itself-iff option.exhaust-sel option.map-sel option.sel
option.sel remove-key-None-iff
dist)

have *helperc1*: ‹*a* ∈# *mset-nodes m'* ⟹ *h* = *m'* ⟹ *find-key a m'* = *None* ⟹ *False*›
by (*metis find-key-None-remove-key-ident in-remove-key-changed option.sel option-hd-Nil option-hd-Some-iff*(*1*))

have *helperc2*: ‹
∀ *x*∈#*mset-nodes m'*. *parents x* = *map-option node* (*hp-parent x m'*) ⟹

hp-parent x′ m′ = map-option (λx. the (remove-key a x)) (hp-parent x′ m′) ⟹
map-option node (hp-parent x′ m′) = map-option (λx. node (the (remove-key a x))) (hp-parent x′ m′)›
by (*metis* (*mono-tags*, *lifting*) *None-eq-map-option-iff map-option-cong option.map-sel option.sel*)
have *helperc3*: *False*
if
‹*remove-key a h = Some m′*› **and**
‹*hp-parent a m′ = Some* (*the* (*remove-key a y*))› **and**
‹*hp-parent a h = Some y*›
for *y* :: ‹(′*a*, *nat*) *hp*›
by (*metis dist that hp-parent-itself hp-parent-remove-key option.sel option.simps*(*2*))

have *helperc4*: ‹*map-option node* (*hp-parent x′ h*) =
map-option node (*map-option* (λ*x*. *the* (*remove-key a x*)) (*hp-parent x′ h*))›
if
‹*remove-key a h = Some m′*› **and**
‹*hp-parent x′ m′ = map-option* (λ*x*. *the* (*remove-key a x*)) (*hp-parent x′ h*)› **and**
‹*hp-next a h = None*› **and**
‹*hp-parent a h = None*› **and**
‹*hp-prev a h = None*›
by (*metis that find-key-None-remove-key-ident find-key-notin no-relative-ancestor-or-notin option.sel option.simps*(*2*) *remove-key-None-iff*)

have *helperc5*: ‹*map-option node* (*hp-parent x′ h*) = *map-option node* (*map-option* (λ*x*. *the* (*remove-key a x*)) (*hp-parent x′ h*))›
if
‹∀ *x*∈#*mset-nodes h*. *parents x = map-option node* (*hp-parent x h*)› **and**
‹*distinct-mset* (*mset-nodes h*)› **and**
‹*node m′* ∈# *mset-nodes h*› **and**
‹*remove-key a h = Some m′*› **and**
‹*hp-parent x′ m′ = map-option* (λ*x*. *the* (*remove-key a x*)) (*hp-parent x′ h*)› **and**
‹*x′* ∉# *the* (*map-option mset-nodes* (*find-key a h*))›
‹*node* (*the* (*None* :: (′*a*, *nat*) *hp option*)) = *x′*›
using *that* **apply** −
apply (*rule map-option-node-map-option-node-iff*)
apply (*meson distinct-mset-hp-parent option.exhaust-sel*)
apply *auto*[]
apply (*smt* (*verit*, *ccfv-threshold*) *Duplicate-Free-Multiset.distinct-mset-mono None-eq-map-option-iff find-key-None-or-itself find-key-None-remove-key-ident hp-child-find-key hp-child-hp-parent hp-parent-hp-child hp-parent-remove-key in-remove-key-changed mset-nodes-find-key-subset node-in-mset-nodes option.map-sel option.sel option-last-Nil option-last-Some-iff*(*2*) *remove-key-notin-unchanged*)
done
have *helperc6*: ‹*map-option node* (*hp-parent x′ h*) = *map-option* (λ*x*. *node* (*the* (*remove-key a x*))) (*hp-parent x′ h*)›
if
‹∀ *x*∈#*mset-nodes h*. *parents x = map-option node* (*hp-parent x h*)› **and**
‹*remove-key a h = Some m′*› **and**
‹*hp-parent x′ m′ = map-option* (λ*x*. *the* (*remove-key a x*)) (*hp-parent x′ h*)› **and**
‹*x′* ∉# *the* (*map-option mset-nodes* (*find-key a h*))›
using *that dist*
by ((*smt* (*verit*, *ccfv-SIG*) *Duplicate-Free-Multiset.distinct-mset-mono None-eq-map-option-iff find-key-None-or-itself find-key-None-remove-key-ident hp-child-find-key hp-child-hp-parent hp-parent-None-notin hp-parent-hp-child map-option-cong mset-nodes-find-key-subset node-in-mset-nodes node-remove-key-itself-iff*

option.map-sel option.sel option-last-Nil option-last-Some-iff(*2*) *remove-key-None-iff*)+)[]
have *helperd1*: ‹*hp-parent a m′* = *None*›
if
‹*a* ∈# *mset-nodes h*› **and**
‹*find-key a h* = *Some m′*› **and**
‹*hp-next a h* = *None*› **and**
‹*hp-parent a h* = *None*› **and**
‹*hp-prev a h* = *None*›
by (*metis that ACIDS.find-key-node-itself no-relative-ancestor-or-notin option.sel*)
have *helperd2*: ‹*hp-parent a m′* = *None*›
if
‹*find-key a h* = *Some m′*›
by (*metis dist that Duplicate-Free-Multiset.distinct-mset-mono find-key-None-or-itself hp-parent-itself mset-nodes-find-key-subset option.sel option.simps*(*3*))
have *helperd3*: ‹*node ya* ∉# *mset-nodes m′*›
if
‹*distinct-mset* (*mset-nodes m′* + *mset-nodes ya*)›
for *ya* :: ‹(*′a*, *nat*) *hp*›
by (*smt* (*verit*, *best*) *that disjunct-not-in distinct-mset-add node-in-mset-nodes option.sel option.simps*(*3*))

show ‹*fst* (*snd* (*snd* (*snd arr′*))) *x′* = *map-option node* (*hp-parent x′ m′*)›
using *parents dist H M′* **apply** −
apply (*frule in-remove-key-in-find-keyD*)
apply (*solves auto*)[]
apply (*solves auto*)[]
unfolding *union-iff*
apply (*rule disjE*, *assumption*)
subgoal
unfolding *assms*(*1*−*5*) *arr*
using *find-key-None-remove-key-ident*[*of a h*]
hp-parent-remove-key-other[*of h a x′*]
distinct-mset-hp-parent[*of h a* ‹*the* (*hp-parent a h*)›]
by (*clarsimp simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def map-option.compositionality comp-def map-option-node-hp-next-remove-key hp-update-parents-def in-the-default-empty-iff*
intro: *helper1*
split: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*)
(*intro conjI impI allI*; *auto dest*: *helper1 helper3*
dest: *helperb4 hp-next-not-same-node*
intro: *helperc1 helperc2 helperc3*
dest: *helperc4 helperc5 intro*: *helperc6*)+
subgoal
unfolding *assms*(*1*−*5*) *arr*
using *in-find-key-same-hp-parent*[*of x′ m′ h a*]
in-find-key-same-hp-parent2[*of x′ m′ h a*]
distinct-mset-find-node-next[*of h a* ‹*the* (*find-key a h*)›]
by (*cases* ‹*x′* = *a*›) (*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def helperd3*
map-option.compositionality comp-def map-option-node-hp-next-remove-key hp-update-parents-def in-the-default-empty-iff
split: *if-splits simp del*: *find-key-None-or-itself hp-parent-itself*
intro: *helperd1 simp*: *helperd2*)
done

show ‹*snd* (*snd* (*snd* (*snd arr′*))) *x′* = *map-option score* (*hp-node x′ m′*)›

using *scores M′ dist H*
hp-child-find-key[of h a x′]
in-remove-key-in-nodes[of a h x′] in-find-key-notin-remove-key[of h a x′]
in-find-key-in-nodes[of a h x′]
hp-parent-hp-child[of h x′] hp-child-hp-parent[of h x′]
hp-node-in-find-key[of h a x′]
unfolding *assms(1−5) arr*
using *hp-score-remove-key-other[of h a x′] find-key-None-or-itself[of a h]*
hp-next-find-key-itself[of h a] has-prev-still-in-remove-key[of h a]
in-remove-key-changed[of a h]
hp-parent-itself[of h] remove-key-None-iff[of a h] find-key-head-node-iff[of h m′]
node-remove-key-in-mset-nodes[of a h]
by (*auto simp add*: *hp-update-child-def hp-update-prev-def hp-update-nxt-def*
map-option.compositionality comp-def map-option-node-hp-next-remove-key hp-update-parents-def
in-the-default-empty-iff
split: *if-splits* *simp del*: *find-key-None-or-itself hp-parent-itself*)
next
fix x :: *′a*
assume ‹$x \in\#\ \sum_{\#}$ (*mset-nodes* ‘# *mset* []))›
then show
‹*fst* (*snd arr′*) x = *map-option node* (*hp-next-children x* []))›
‹*fst arr′* x = *map-option node* (*hp-prev-children x* []))›
‹*fst* (*snd* (*snd arr′*)) x = *map-option node* (*hp-child-children x* []))› **and**
‹*fst* (*snd* (*snd* (*snd arr′*))) x = *map-option node* (*hp-parent-children x* []))›
‹*snd* (*snd* (*snd* (*snd arr′*))) x = *map-option score* (*hp-node-children x* []))›
by *auto*
next
have *H*: ‹($\sum_{\#}$ (*mset-nodes* ‘#
((*if remove-key a h = None then* {#} *else* {#*the* (*remove-key a h*)#}) +
(*if find-key a h = None then* {#} *else* {#*the* (*find-key a h*)#})) +
mset-nodes ‘# *mset* [])) = ($\sum_{\#}$(*mset-nodes* ‘# {#*h*#}))›
using *find-remove-mset-nodes-full[of h a ‹the (remove-key a h)› ‹the (find-key a h)›] find-key-None-remove-key-ident[of a h]*
dist
apply (*cases* ‹*find-key a h*›; *cases* ‹*remove-key a h*›; *auto simp*: *ac-simps*)
apply (*metis find-key-head-node-iff option.sel remove-key-None-iff*)
done
show ‹*empty-outside* ($\sum_{\#}$ (*mset-nodes* ‘# *?a* + *mset-nodes* ‘# *mset* []))
(*fst arr′*)›
using *empty-outside hp-next-in-nodes2[of a h]* **unfolding** *H*
unfolding *assms(1−5) arr* **by** (*auto simp*: *hp-update-parents-def hp-update-prev-def hp-update-child-def*
hp-update-nxt-def empty-outside-alt-def)
show ‹*empty-outside* ($\sum_{\#}$ (*mset-nodes* ‘# *?a* + *mset-nodes* ‘# *mset* []))
(*fst* (*snd* (*snd* (*snd arr′*))))›
using *empty-outside hp-next-in-nodes2[of a h]* **unfolding** *H*
unfolding *assms(1−5) arr* **by** (*auto simp*: *hp-update-parents-def hp-update-prev-def hp-update-child-def*
hp-update-nxt-def empty-outside-alt-def)
qed
qed

In the kissat implementation prev and parent are merged.

lemma *in-node-iff-prev-parent-or-root*:
assumes ‹*distinct-mset* (*mset-nodes h*)›
shows ‹$i \in\#$ *mset-nodes h* ⟷ *hp-prev i h* ≠ *None* ∨ *hp-parent i h* ≠ *None* ∨ *i* = *node h*›
using *assms*
proof (*induction h arbitrary*: *i*)

case (*Hp x1a x2a x3a*) **note** *IH* = *this*(*1*) **and** *dist* = *this*(*2*)
have *?case* **if** *pre*:‹*i* ≠ *x1a*› ‹*i* ∈# *sum-list* (*map mset-nodes x3a*)›
proof −
obtain *c* **where**
c: ‹*c* ∈ *set x3a*› **and**
i-c: ‹*i* ∈#*mset-nodes c*›
using *pre*
unfolding *in-mset-sum-list-iff*
by *auto*
have *dist-c*: ‹*distinct-mset* (*mset-nodes c*)›
using *c dist* **by** (*simp add*: *distinct-mset-add sum-list-map-remove1*)

obtain *ys zs* **where** *x3a-def*: ‹*x3a* = *ys* @ *c* # *zs*›
using *split-list*[*OF c*] **by** *auto*
have *i-ys*: ‹*i* ∉# $\sum_{\#}$ (*mset-nodes* '# *mset ys*)› ‹*i* ∉# *sum-list* (*map mset-nodes zs*)›
using *dist i-c*
by (*auto simp*: *x3a-def disjunct-not-in distinct-mset-add*)
have *dist-c-zs*: ‹*distinct-mset* (*mset-nodes c* + *sum-list* (*map mset-nodes zs*))›
using *WB-List-More.distinct-mset-union2 dist x3a-def* **by** *auto*
consider
‹*i* = *node c*› |
‹*i* ≠ *node c*›
by *blast*
then show *?case*
proof *cases*
case *2*
then have ‹*hp-prev i c* ≠ *None* ⟹ *hp-prev-children i x3a* ≠ *None*›
using *c dist i-c i-ys dist-c-zs* **by** (*auto simp*: *x3a-def hp-prev-children-skip-last-append*[*of* - ‹[-]›, *simplified*])
moreover have ‹*hp-parent i c* ≠ *None* ⟹ *hp-parent-children i x3a* ≠ *None*›
using *c dist i-c* **by** (*auto dest*!: *split-list simp*: *hp-parent-children-append-case hp-parent-children-cons split*: *option.splits*)
ultimately show *?thesis*
using *i-c 2 IH*[*of c i*, *OF c dist-c*]
by (*cases* ‹*hp-prev i c*›)
(*auto simp del*: *hp-prev-None-notin hp-parent-None-notin simp*: *hp-parent-simps-single-if*)
next
case *1*
have ‹*hp-prev-children* (*node c*) (*ys* @ *c* # *zs*) = (*option-last ys*)›
using *i-ys hp-prev-children-Cons-append-found*[*of i ys* ‹*hps c*› *zs* ‹*score c*›] *1 dist-c*
by (*cases c*) (*auto simp del*: *hp-prev-children-Cons-append-found*)
then show *?thesis*
using *c dist i-c i-ys dist-c-zs* **by** (*auto dest*!: *simp*: *x3a-def 1*)
qed
qed
then show *?case*
using *dist IH*
by (*auto simp add*: *hp-parent-none-children*)
qed

lemma *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*:
assumes ‹*encoded-hp-prop-list-conc arr h*› **and** ‹*snd h* ≠ *None*›
shows ‹*i* ∈# *mset-nodes* (*the* (*snd h*)) ⟷ *hp-read-prev i* (*fst* (*snd arr*)) ≠ *None* ∨ *hp-read-parent i* (*fst* (*snd arr*)) ≠ *None* ∨ *Some i* = *snd* (*snd arr*)›
using *assms in-node-iff-prev-parent-or-root*[*of* ‹*the* (*snd h*)› *i*]
by (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def empty-outside-def*

simp del: *hp-prev-None-notin hp-parent-None-notin*)

fun *update-source-node* **where**
‹*update-source-node i* ($\mathcal{V}$,*arr*,-) = ($\mathcal{V}$, *arr*, *i*)›
fun *source-node* :: ‹(*nat multiset* × (*nat*,′*c*) *hp-fun* × *nat option*) ⇒ -› **where**
‹*source-node* ($\mathcal{V}$,*arr*,*h*) = *h*›
fun *hp-read-nxt*′ :: ‹-› **where**
‹*hp-read-nxt*′ *i* ($\mathcal{V}$, *arr*, *h*) = *hp-read-nxt i arr*›
fun *hp-read-parent*′ :: ‹-› **where**
‹*hp-read-parent*′ *i* ($\mathcal{V}$, *arr*, *h*) = *hp-read-parent i arr*›

fun *hp-read-score*′ :: ‹-› **where**
‹*hp-read-score*′ *i* ($\mathcal{V}$, *arr*, *h*) = (*hp-read-score i arr*)›
fun *hp-read-child*′ :: ‹-› **where**
‹*hp-read-child*′ *i* ($\mathcal{V}$, *arr*, *h*) = *hp-read-child i arr*›

fun *hp-read-prev*′ :: ‹-› **where**
‹*hp-read-prev*′ *i* ($\mathcal{V}$, *arr*, *h*) = *hp-read-prev i arr*›

fun *hp-update-child*′ **where**
‹*hp-update-child*′ *i p*($\mathcal{V}$, *u*, *h*) = ($\mathcal{V}$, *hp-update-child i p u*, *h*)›

fun *hp-update-parents*′ **where**
‹*hp-update-parents*′ *i p*($\mathcal{V}$, *u*, *h*) = ($\mathcal{V}$, *hp-update-parents i p u*, *h*)›

fun *hp-update-prev*′ **where**
‹*hp-update-prev*′ *i p* ($\mathcal{V}$, *u*, *h*) = ($\mathcal{V}$, *hp-update-prev i p u*, *h*)›

fun *hp-update-nxt*′ **where**
‹*hp-update-nxt*′ *i p*($\mathcal{V}$, *u*, *h*) = ($\mathcal{V}$, *hp-update-nxt i p u*, *h*)›

fun *hp-update-score*′ **where**
‹*hp-update-score*′ *i p*($\mathcal{V}$, *u*, *h*) = ($\mathcal{V}$, *hp-update-score i p u*, *h*)›

definition *maybe-hp-update-prev*′ **where**
‹*maybe-hp-update-prev*′ *child ch arr* =
(*if child* = *None then arr else hp-update-prev*′ (*the child*) *ch arr*)›

definition *maybe-hp-update-nxt*′ **where**
‹*maybe-hp-update-nxt*′ *child ch arr* =
(*if child* = *None then arr else hp-update-nxt*′ (*the child*) *ch arr*)›

definition *maybe-hp-update-parents*′ **where**
‹*maybe-hp-update-parents*′ *child ch arr* =
(*if child* = *None then arr else hp-update-parents*′ (*the child*) *ch arr*)›

definition *maybe-hp-update-child*′ **where**
‹*maybe-hp-update-child*′ *child ch arr* =
(*if child* = *None then arr else hp-update-child*′ (*the child*) *ch arr*)›

definition *unroot-hp-tree* **where**
‹*unroot-hp-tree arr h* = *do* {
ASSERT (*h* ∈# *fst arr*);
let a = *source-node arr*;

ASSERT (*a* ≠ *None* ⟶ *the a* ∈# *fst arr*);
let nnext = *hp-read-nxt′ h arr*;
let parent = *hp-read-parent′ h arr*;
let prev = *hp-read-prev′ h arr*;
if prev = *None* ∧ *parent* = *None* ∧ *Some h* ≠ *a then RETURN* (*update-source-node None arr*)
else if Some h = *a then RETURN* (*update-source-node None arr*)
else do {
ASSERT (*a* ≠ *None*);
ASSERT (*nnext* ≠ *None* ⟶ *the nnext* ∈# *fst arr*);
ASSERT (*parent* ≠ *None* ⟶ *the parent* ∈# *fst arr*);
ASSERT (*prev* ≠ *None* ⟶ *the prev* ∈# *fst arr*);
let a′ = *the a*;
let arr = *maybe-hp-update-child′ parent nnext arr*;
let arr = *maybe-hp-update-nxt′ prev nnext arr*;
let arr = *maybe-hp-update-prev′ nnext prev arr*;
let arr = *maybe-hp-update-parents′ nnext parent arr*;

let arr = *hp-update-nxt′ h None arr*;
let arr = *hp-update-prev′ h None arr*;
let arr = *hp-update-parents′ h None arr*;

let arr = *hp-update-nxt′ h* (*Some a′*) *arr*;
let arr = *hp-update-prev′ a′* (*Some h*) *arr*;
RETURN (*update-source-node None arr*)
}
}›

lemma *unroot-hp-tree-alt-def*:
‹*unroot-hp-tree arr h* = *do* {
ASSERT (*h* ∈# *fst arr*);
let a = *source-node arr*;
ASSERT (*a* ≠ *None* ⟶ *the a* ∈# *fst arr*);
let nnext = *hp-read-nxt′ h arr*;
let parent = *hp-read-parent′ h arr*;
let prev = *hp-read-prev′ h arr*;
if prev = *None* ∧ *parent* = *None* ∧ *Some h* ≠ *a then RETURN* (*update-source-node None arr*)
else if Some h = *a then RETURN* (*update-source-node None arr*)
else do {
ASSERT (*a* ≠ *None*);
ASSERT (*nnext* ≠ *None* ⟶ *the nnext* ∈# *fst arr*);
ASSERT (*parent* ≠ *None* ⟶ *the parent* ∈# *fst arr*);
ASSERT (*prev* ≠ *None* ⟶ *the prev* ∈# *fst arr*);
let a′ = *the a*;
arr ← *do* {
let arr = *maybe-hp-update-child′ parent nnext arr*;
let arr = *maybe-hp-update-nxt′ prev nnext arr*;
let arr = *maybe-hp-update-prev′ nnext prev arr*;
let arr = *maybe-hp-update-parents′ nnext parent arr*;

let arr = *hp-update-nxt′ h None arr*;
let arr = *hp-update-prev′ h None arr*;
let arr = *hp-update-parents′ h None arr*;

RETURN (*update-source-node None arr*)
};

let arr = hp-update-nxt′ h (Some a′) arr;
let arr = hp-update-prev′ a′ (Some h) arr;
RETURN (arr)
}
}›
unfolding *unroot-hp-tree-def nres-monad3 Let-def*
apply (*cases arr*)
by (*auto intro*!: *bind-cong*[*OF refl*] *simp*: *maybe-hp-update-child′-def*
maybe-hp-update-nxt′-def maybe-hp-update-prev′-def maybe-hp-update-parents′-def)

lemma *hp-update-fst-snd*:
‹*hp-update-prev i j* (*fst* (*snd arr*)) = *fst* (*snd* (*hp-update-prev′ i j arr*))›
‹*hp-update-nxt i j* (*fst* (*snd arr*)) = *fst* (*snd* (*hp-update-nxt′ i j arr*))›
‹*hp-update-parents i j* (*fst* (*snd arr*)) = *fst* (*snd* (*hp-update-parents′ i j arr*))›
‹*hp-update-child i j* (*fst* (*snd arr*)) = *fst* (*snd* (*hp-update-child′ i j arr*))›
by (*solves* ‹*cases arr*; *auto*›)+

lemma *find-remove-mset-nodes-full2*:
‹*distinct-mset* (*mset-nodes h*) ⟹ *sum-mset* (*mset-nodes* ‘# ((*if remove-key a h* = *None then* {#} *else* {#*the* (*remove-key a h*)#}) +
(*if find-key a h* = *None then* {#} *else* {#*the* (*find-key a h*)#}))) = *mset-nodes h*›
using *find-remove-mset-nodes-full*[*of h a*]
apply (*auto*)
apply (*auto simp add*: *find-key-None-remove-key-ident*)
apply (*metis find-key-head-node-iff option.sel remove-key-None-iff*)
done

definition *encoded-hp-prop-mset2-conc*
:: ′*a*::*linorder multiset* × (′*a*, ′*b*) *hp-fun* × ′*a option* ⇒
′*a*::*linorder multiset* × (′*a*, ′*b*) *hp multiset* ⇒ *bool*
where
‹*encoded-hp-prop-mset2-conc* = (λ($\mathcal{V}$, *arr*, *h*) ($\mathcal{V}$′, *x*). $\mathcal{V}$ = $\mathcal{V}$′ ∧
(*encoded-hp-prop-list* $\mathcal{V}$ *x* [] *arr*))›

lemma *fst-update*[*simp*]:
‹ *fst* (*hp-update-prev′ a b x*) = *fst x*›
‹*fst* (*hp-update-nxt′ a b x*) = *fst x*›
‹*fst* (*update-source-node y x*) = *fst x*›
by (*cases x*;*auto*; *fail*)+

lemma *encoded-hp-prop-mset2-conc-combine-list2-conc*:
‹*encoded-hp-prop-mset2-conc arr* ($\mathcal{V}$, {#*a*,*b*#}) ⟹
encoded-hp-prop-list2-conc (*hp-update-prev′* (*node b*) (*Some* (*node a*)) (*hp-update-nxt′* (*node a*) (*Some* (*node b*)) (*update-source-node None arr*))) ($\mathcal{V}$, [*a*,*b*])›
unfolding *encoded-hp-prop-mset2-conc-def encoded-hp-prop-list2-conc-alt-def*
encoded-hp-prop-list-def case-prod-beta
apply (*intro conjI*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal

apply (*cases arr*)
apply (*auto simp*: *encoded-hp-prop-list-def hp-update-prev-def hp-update-nxt-def*)
apply (*metis hp-next-None-notin hp-next-children.simps*(*2*) *hp-next-children-simps*(*2*) *hp-next-children-simps*(*3*))
by (*metis hp-next-None-notin hp-next-children.simps*(*2*) *hp-next-children-simps*(*2*) *hp-next-children-simps*(*3*))
subgoal
apply (*cases arr*)
apply (*auto simp*: *encoded-hp-prop-list-def hp-update-prev-def hp-update-nxt-def*)
apply (*metis hp-prev-None-notin hp-prev-children.simps*(*2*) *hp-prev-children-simps*(*2*) *hp-prev-children-simps*(*3*))
by (*metis hp-prev-None-notin hp-prev-children.simps*(*2*) *hp-prev-children-simps*(*2*) *hp-prev-children-simps*(*3*))
subgoal
apply (*cases arr*)
apply (*auto simp*: *encoded-hp-prop-list-def hp-update-prev-def hp-update-nxt-def*)
by (*metis hp-child-None-notin hp-child-children-hp-child hp-child-children-simps*(*2*) *hp-child-children-simps*(*3*))+
subgoal
apply (*cases arr*)
apply (*auto simp*: *encoded-hp-prop-list-def hp-update-prev-def hp-update-nxt-def*)
by (*metis hp-parent-None-notin hp-parent-children-cons hp-parent-children-single-hp-parent option.case-eq-if*)
subgoal
apply (*cases arr*)
apply (*auto simp*: *encoded-hp-prop-list-def hp-update-prev-def hp-update-nxt-def*)
by (*metis hp-node-None-notin2 hp-node-children-Cons-if*)
subgoal
by (*cases arr*)
(*auto simp*: *encoded-hp-prop-list-def hp-update-prev-def hp-update-nxt-def*)
subgoal
by (*cases arr*)
(*auto simp*: *encoded-hp-prop-list-def hp-update-prev-def hp-update-nxt-def*)
subgoal
by (*cases arr*)
(*auto simp*: *encoded-hp-prop-list-def hp-update-prev-def hp-update-nxt-def*)
done

lemma *update-source-node-fst-simps*[*simp*]:
‹*fst* (*snd* (*update-source-node None arr*)) = *fst* (*snd arr*)›
‹*fst* (*update-source-node None arr*) = *fst arr*›
‹*snd* (*snd* (*update-source-node None arr*)) = *None*›
by (*solves* ‹*cases arr*; *auto*›)+

lemma *maybe-hp-update-fst-snd*: ‹*fst* (*snd* (*maybe-hp-update-child′* (*map-option node b*) *x arr*)) =
(*if b* = *None then fst* (*snd arr*) *else fst* (*snd* (*hp-update-child′* (*node* (*the b*)) *x arr*)))›
‹*fst* (*snd* (*maybe-hp-update-prev′* (*map-option node b*) *x arr*)) =
(*if b* = *None then fst* (*snd arr*) *else fst* (*snd* (*hp-update-prev′* (*node* (*the b*)) *x arr*)))›
‹*fst* (*snd* (*maybe-hp-update-nxt′* (*map-option node b*) *x arr*)) =
(*if b* = *None then fst* (*snd arr*) *else fst* (*snd* (*hp-update-nxt′* (*node* (*the b*)) *x arr*)))›
‹*fst* (*snd* (*maybe-hp-update-parents′* (*map-option node b*) *x arr*)) =
(*if b* = *None then fst* (*snd arr*) *else fst* (*snd* (*hp-update-parents′* (*node* (*the b*)) *x arr*)))› **and**
maybe-hp-update-fst-snd2:
‹((*maybe-hp-update-child′* (*map-option node b*) *x arr′*)) =
(*if b* = *None then* (*arr′*) *else* ((*hp-update-child′* (*node* (*the b*)) *x arr′*)))›
‹((*maybe-hp-update-prev′* (*map-option node b*) *x arr′*)) =
(*if b* = *None then* (*arr′*) *else* ((*hp-update-prev′* (*node* (*the b*)) *x arr′*)))›
‹((*maybe-hp-update-nxt′* (*map-option node b*) *x arr′*)) =
(*if b* = *None then* (*arr′*) *else* ((*hp-update-nxt′* (*node* (*the b*)) *x arr′*)))›
‹((*maybe-hp-update-parents′* (*map-option node b*) *x arr′*)) =
(*if b* = *None then* (*arr′*) *else* ((*hp-update-parents′* (*node* (*the b*)) *x arr′*)))›
for *x b arr*

apply (*solves* ‹*cases arr*; *auto simp*: *maybe-hp-update-child′-def maybe-hp-update-parents′-def maybe-hp-update-prev′-def maybe-hp-update-nxt′-def maybe-hp-update-prev′-def maybe-hp-update-nxt′-def*›)+
done

lemma *fst-hp-update-simp*[*simp*]:
‹*fst* (*hp-update-prev′ i x arr*) = *fst arr*›
‹*fst* (*hp-update-nxt′ i x arr*) = *fst arr*›
‹*fst* (*hp-update-child′ i x arr*) = *fst arr*›
‹*fst* (*hp-update-parents′ i x arr*) = *fst arr*›
by (*solves* ‹*cases arr*; *auto*›)+

lemma *fst-maybe-hp-update-simp*[*simp*]:
‹*fst* (*maybe-hp-update-prev′ i y arr*) = *fst arr*›
‹*fst* (*maybe-hp-update-nxt′ i y arr*) = *fst arr*›
‹*fst* (*maybe-hp-update-child′ i y arr*) = *fst arr*›
‹*fst* (*maybe-hp-update-parents′ i y arr*) = *fst arr*›
by (*solves* ‹*cases arr*; *cases i*; *auto simp*: *maybe-hp-update-prev′-def maybe-hp-update-nxt′-def maybe-hp-update-child′-def maybe-hp-update-parents′-def*›)+

lemma *encoded-hp-prop-list-remove-find2*:
fixes *h* :: ‹(*′a*::*linorder*, *nat*) *hp*› **and** *a arr* **and** *hs* :: ‹(*′a*, *nat*) *hp multiset*›
defines ‹arr_1 ≡ (*if hp-parent a h* = *None then arr else hp-update-child′* (*node* (*the* (*hp-parent a h*))) (*map-option node* (*hp-next a h*)) *arr*)›
defines ‹arr_2 ≡ (*if hp-prev a h* = *None then* arr_1 *else hp-update-nxt′* (*node* (*the* (*hp-prev a h*))) (*map-option node* (*hp-next a h*)) arr_1)›
defines ‹arr_3 ≡ (*if hp-next a h* = *None then* arr_2 *else hp-update-prev′* (*node* (*the* (*hp-next a h*))) (*map-option node* (*hp-prev a h*)) arr_2)›
defines ‹arr_4 ≡ (*if hp-next a h* = *None then* arr_3 *else hp-update-parents′* (*node* (*the* (*hp-next a h*))) (*map-option node* (*hp-parent a h*)) arr_3)›
defines ‹*arr′* ≡ *hp-update-parents′ a None* (*hp-update-prev′ a None* (*hp-update-nxt′ a None* arr_4))›
assumes *enc*: ‹*encoded-hp-prop-mset2-conc arr* (𝒱, *add-mset h* {#})›
shows ‹*encoded-hp-prop-mset2-conc arr′* (𝒱, (*if remove-key a h* = *None then* {#} *else* {#*the* (*remove-key a h*)#}) +
(*if find-key a h* = *None then* {#} *else* {#*the* (*find-key a h*)#}))›
using *encoded-hp-prop-list-remove-find*[*of* 𝒱 *h* ‹*fst* (*snd arr*)› *a*] *enc*
unfolding *assms*(*1*−*5*) **apply** −
unfolding *encoded-hp-prop-mset2-conc-def case-prod-beta hp-update-fst-snd*
apply (*subst hp-update-fst-snd*[*symmetric*])
apply (*subst hp-update-fst-snd*[*symmetric*])
apply (*subst hp-update-fst-snd*[*symmetric*])
unfolding *maybe-hp-update-fst-snd*[*symmetric*] *maybe-hp-update-parents′-def*[*symmetric*] *maybe-hp-update-nxt′-def*[*symmetric*] *maybe-hp-update-prev′-def*[*symmetric*] *maybe-hp-update-child′-def*[*symmetric*] *encoded-hp-prop-mset2-conc-def case-prod-beta hp-update-fst-snd maybe-hp-update-fst-snd2*[*symmetric*] *maybe-hp-update-fst-snd*[*symmetric*]
by *auto*

lemma *hp-read-fst-snd-simps*[*simp*]:
‹*hp-read-nxt j* (*fst* (*snd arr*)) = *hp-read-nxt′ j arr*›
‹*hp-read-prev j* (*fst* (*snd arr*)) = *hp-read-prev′ j arr*›
‹*hp-read-child j* (*fst* (*snd arr*)) = *hp-read-child′ j arr*›
‹*hp-read-parent j* (*fst* (*snd arr*)) = *hp-read-parent′ j arr*›
‹*hp-read-score j* (*fst* (*snd arr*)) = *hp-read-score′ j arr*›
by (*solves* ‹*cases arr*; *auto*›)+

lemma *unroot-hp-tree*:
fixes *h* :: ‹(*nat*, *nat*)*hp option*›
assumes *enc*: ‹*encoded-hp-prop-list-conc arr* (𝒱, *h*)› ‹*a* ∈# *fst arr*› ‹*h* ≠ *None*›
shows ‹*unroot-hp-tree arr a* ≤ *SPEC* (λ*arr*′. *fst arr*′ = *fst arr* ∧ *encoded-hp-prop-list2-conc arr*′
(𝒱, (*if find-key a* (*the h*) = *None then* [] *else* [*the* (*find-key a* (*the h*))]) @
(*if remove-key a* (*the h*) = *None then* [] *else* [*the* (*remove-key a* (*the h*))]))))›
proof −
obtain *prevs nxts childs parents scores k* **where**
arr: ‹*arr* = (𝒱, (*prevs*, *nxts*, *childs*, *parents*, *scores*), *k*)› **and**
dist: ‹*distinct-mset* (*mset-nodes* (*the h*))› **and**
k: ‹*k* = *Some* (*node* (*the h*))›‹*the k* ∈# 𝒱› **and**
𝒱: ‹*set-mset* ((*mset-nodes* (*the h*))) ⊆ *set-mset* 𝒱›
by (*cases arr*; *cases* ‹*the h*›) (*use assms* **in** ‹*auto simp*: *ac-simps encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def*
encoded-hp-prop-list-conc-def encoded-hp-prop-def›)
have *K1*: ‹*fst* (*snd* (*maybe-hp-update-child*′ (*map-option node b*) *x arr*)) =
(*if b* = *None then fst* (*snd arr*) *else fst* (*snd* (*hp-update-child*′ (*node* (*the b*)) *x arr*)))›
‹*fst* (*snd* (*maybe-hp-update-prev*′ (*map-option node b*) *x arr*)) =
(*if b* = *None then fst* (*snd arr*) *else fst* (*snd* (*hp-update-prev*′ (*node* (*the b*)) *x arr*)))›
‹*fst* (*snd* (*maybe-hp-update-nxt*′ (*map-option node b*) *x arr*)) =
(*if b* = *None then fst* (*snd arr*) *else fst* (*snd* (*hp-update-nxt*′ (*node* (*the b*)) *x arr*)))›
‹*fst* (*snd* (*maybe-hp-update-parents*′ (*map-option node b*) *x arr*)) =
(*if b* = *None then fst* (*snd arr*) *else fst* (*snd* (*hp-update-parents*′ (*node* (*the b*)) *x arr*)))›
for *x b arr*
apply (*solves* ‹*cases arr*; *auto simp*: *maybe-hp-update-child*′*-def maybe-hp-update-parents*′*-def*
maybe-hp-update-prev′*-def maybe-hp-update-nxt*′*-def maybe-hp-update-prev*′*-def*
maybe-hp-update-nxt′*-def*›)+
done
have *source-node-alt*: ‹*snd* (*snd arr*) = *source-node arr*›
by (*cases arr*) *auto*
have *KK*: ‹*a*∈#*mset-nodes* (*the h*) ⟹ *nxts a* = *map-option node* (*hp-next a* (*the h*))›
‹*a*∈#*mset-nodes* (*the h*) ⟹ *prevs a* = *map-option node* (*hp-prev a* (*the h*))›
‹*a*∈#*mset-nodes* (*the h*) ⟹ *parents a* = *map-option node* (*hp-parent a* (*the h*))›
‹*a*∈#*mset-nodes* (*the h*) ⟹ *childs a* = *map-option node* (*hp-child a* (*the h*))›
using *enc*
unfolding *arr encoded-hp-prop-list-conc-def*
by (*auto simp*: *encoded-hp-prop-def*)
have *KK*′: ‹*a*∈#*mset-nodes* (*the h*) ⟹ *nxts a* ≠ *None* ⟹ *the* (*nxts a*) ∈# 𝒱›
‹*a*∈#*mset-nodes* (*the h*) ⟹ *prevs a* ≠ *None* ⟹ *the* (*prevs a*) ∈# 𝒱›
‹*a*∈#*mset-nodes* (*the h*) ⟹ *parents a* ≠ *None* ⟹ *the* (*parents a*) ∈# 𝒱›
‹*a*∈#*mset-nodes* (*the h*) ⟹ *childs a* ≠ *None* ⟹ *the* (*childs a*) ∈# 𝒱›
using *enc* 𝒱 *KK hp-next-in-nodes2*[*of a* ‹*the h*› ‹*the* (*hp-next a* (*the h*))›] *dist*
hp-parent-None-notin[*of a* ‹*the h*›]
hp-prev-in-nodes[*of a* ‹*the h*›]
hp-parent-in-nodes[*of a* ‹*the h*›]
hp-parent-hp-child[*of* ‹*the h*› *a*]
unfolding *arr encoded-hp-prop-list-conc-def*
apply (*auto simp*: *encoded-hp-prop-def*)
by (*metis hp-parent-None-notin mset-set-set-mset-msubset mset-subset-eqD option.simps*(*3*))
have *KK2*: ‹*fst* (*hp-update-parents*′ *a None*
(*hp-update-prev*′ *a None*
(*hp-update-nxt*′ *a None*
(*maybe-hp-update-parents*′ (*nxts a*) (*parents a*)
(*maybe-hp-update-prev*′ (*nxts a*) (*Some* (*node z*))
(*maybe-hp-update-nxt*′ (*Some* (*node z*)) (*nxts a*)
(*maybe-hp-update-child*′ (*parents a*) (*nxts a*)

(𝒱, (*prevs*, *nxts*, *childs*, *parents*, *scores*), *Some* (*node y*))))))))) = 𝒱›
by *auto*
have *HH*: ‹*encoded-hp-prop-list* 𝒱 {#*the h*#} [] (*fst* (*snd* (*arr*)))› ‹*encoded-hp-prop-mset2-conc arr* (𝒱, {#*the h*#})›
using *assms* **unfolding** *encoded-hp-prop-list-def encoded-hp-prop-list-conc-def*
encoded-hp-prop-mset2-conc-def
by *auto*
have *KK3*: ‹*a*∈#*mset-nodes* (*the h*) ⟹ *remove-key a* (*the h*) = *None* ∨ *node* (*the* (*remove-key a* (*the h*))) = *node* (*the h*)›
by (*cases* ‹*the h*›; *auto simp*:)
let *?arr* = ‹*hp-update-parents′ a None*
(*hp-update-prev′ a None*
(*hp-update-nxt′ a None*
(*maybe-hp-update-parents′* (*map-option node* (*hp-next a* (*the h*)))
(*map-option node* (*hp-parent a* (*the h*)))
(*maybe-hp-update-prev′* (*map-option node* (*hp-next a* (*the h*))) (*map-option node* (*hp-prev a* (*the h*)))
(*maybe-hp-update-nxt′* (*map-option node* (*hp-prev a* (*the h*)))
(*map-option node* (*hp-next a* (*the h*)))
(*maybe-hp-update-child′* (*map-option node* (*hp-parent a* (*the h*)))
(*map-option node* (*hp-next a* (*the h*))) *arr*))))))›
have *update-source-node-None-alt*: ‹*update-source-node None x* = (*fst x*, *fst* (*snd x*), *None*)› **for** *x*
by (*cases x*) *auto*
show *?thesis*
using *assms*
unfolding *unroot-hp-tree-alt-def*
apply *refine-vcg*
subgoal using *k* **unfolding** *arr* **by** *auto*
subgoal using *k* **unfolding** *arr* **by** *auto*
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr* ‹(𝒱, *h*)› *a*]
unfolding *source-node-alt*
by (*auto simp add*: *find-key-None-remove-key-ident encoded-hp-prop-mset2-conc-def arr*)
(*solves* ‹*auto simp*: *encoded-hp-prop-list2-conc-def encoded-hp-prop-list-conc-def*›)+
subgoal using *k* **unfolding** *arr* **by** *auto*
subgoal
unfolding
hp-update-fst-snd K1[*symmetric*] *arr encoded-hp-prop-list-conc-def encoded-hp-prop-mset2-conc-def*
by (*auto simp*: *remove-key-None-iff encoded-hp-prop-list2-conc-def*)
subgoal
unfolding
hp-update-fst-snd K1[*symmetric*] *arr encoded-hp-prop-list-conc-def encoded-hp-prop-mset2-conc-def*
by *clarsimp*
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr* ‹(𝒱, *h*)› *a*] *KK′* **unfolding** *arr* **by** *auto*
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr* ‹(𝒱, *h*)› *a*] *KK′* **unfolding** *arr* **by** *auto*
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr* ‹(𝒱, *h*)› *a*] *KK′* **unfolding** *arr* **by** *auto*
subgoal using *k* **unfolding** *arr* **by** *auto*
subgoal
apply ((*split if-splits*)+; *intro impI conjI*)
subgoal by (*simp add*: *find-key-None-remove-key-ident*)
subgoal

using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr* ‹($\mathcal{V}$, *h*)› *a*] *KK′* **unfolding** *arr*
apply (*simp add*: *find-key-None-remove-key-ident arr*)
by (*metis find-key-None-remove-key-ident in-remove-key-changed option.sel option.simps*(*3*))
subgoal
by (*auto simp*: *remove-key-None-iff encoded-hp-prop-list2-conc-def encoded-hp-prop-list-conc-def*)
subgoal
unfolding *append.append-Cons append.append-Nil*
apply (*insert encoded-hp-prop-list-remove-find2*[*of* ‹*arr*› $\mathcal{V}$ ‹*the h*› *a*, *OF HH*(*2*)])
unfolding *K1*[*symmetric*]
maybe-hp-update-child′-def[*symmetric*] *maybe-hp-update-parents′-def*[*symmetric*]
maybe-hp-update-prev′-def[*symmetric*] *maybe-hp-update-nxt′-def*[*symmetric*]
hp-update-fst-snd
unfolding *maybe-hp-update-fst-snd*[*symmetric*] *maybe-hp-update-parents′-def*[*symmetric*]
maybe-hp-update-nxt′-def[*symmetric*] *maybe-hp-update-prev′-def*[*symmetric*] *maybe-hp-update-child′-def*[*symmetric*]
hp-update-fst-snd maybe-hp-update-fst-snd2[*symmetric*]
maybe-hp-update-fst-snd[*symmetric*]
apply (*subst arg-cong*[*of - -* ‹λ*arr. encoded-hp-prop-list2-conc arr -*›])
defer
apply (*rule encoded-hp-prop-mset2-conc-combine-list2-conc*[*of ?arr* $\mathcal{V}$ ‹*the* (*find-key a* (*the h*))› ‹*the* (*remove-key a* (*the h*))›])
subgoal using *HH*(*2*) **by** (*auto simp*: *add-mset-commute*)
subgoal using *KK*[*symmetric*] *KK3*
encoded-hp-prop-list-in-node-iff-prev-parent-or-root[*of* ‹*arr*› ‹($\mathcal{V}$, *h*)› *a*] *arr k*
by *auto*
done
done
done
qed

definition *rescale-and-reroot* **where**
‹*rescale-and-reroot h w′ arr = do* {
ASSERT (*h* ∈# *fst arr*);
let nnext = hp-read-nxt′ h arr;
let parent = hp-read-parent′ h arr;
let prev = hp-read-prev′ h arr;
if source-node arr = None then RETURN (*hp-update-score′ h* (*Some w′*) *arr*)
else if prev = None ∧ *parent = None* ∧ *Some h* ≠ *source-node arr then RETURN* (*hp-update-score′ h* (*Some w′*) *arr*)
else if Some h = source-node arr then RETURN (*hp-update-score′ h* (*Some w′*) *arr*)
else do {
arr ← *unroot-hp-tree arr h*;
ASSERT (*h* ∈# *fst arr*);
let arr = (*hp-update-score′ h* (*Some w′*) *arr*);
merge-pairs arr h
}
}›

lemma *fst-update2*[*simp*]:
‹*fst* (*hp-update-score′ a b h*) = *fst h*›
by (*cases h*; *auto*; *fail*)+

lemma *encoded-hp-prop-list2-conc-update-score*:
‹*encoded-hp-prop-list2-conc h* ($\mathcal{V}$, [*x*,*y*]) ⟹ *node x = a* ⟹ *encoded-hp-prop-list2-conc* (*hp-update-score′ a* (*Some w′*) *h*) ($\mathcal{V}$, [*Hp* (*node x*) *w′* (*hps x*), *y*])›
unfolding *encoded-hp-prop-list2-conc-alt-def case-prod-beta encoded-hp-prop-list-def*
apply (*intro conjI conjI allI impI ballI*)

subgoal by *auto*
subgoal by (*cases x*) *auto*
subgoal by (*cases x*) *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
apply (*cases x*; *cases h*)
apply (*clarsimp simp add*: *encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def hp-update-score-def*)
by (*smt* (*verit*, *ccfv-threshold*) *add-diff-cancel-left′ add-diff-cancel-right′ distinct-mset-in-diff hp.sel*(*1*) *hp-next-children.simps*(*2*)
hp-next-children-simps(*1*) *hp-next-children-simps*(*2*) *hp-next-children-simps*(*3*) *hp-next-simps option.map*(*2*) *set-mset-union*)
subgoal
apply (*cases x*; *cases h*)
apply (*clarsimp simp add*: *encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def hp-update-score-def*)
by (*metis* (*no-types*, *lifting*) *None-eq-map-option-iff Un-iff hp.sel*(*1*) *hp-prev-None-notin hp-prev-None-notin-children hp-prev-children.simps*(*2*) *hp-prev-children-simps hp-prev-simps node-in-mset-nodes option.map*(*2*))
subgoal
apply (*cases x*; *cases h*)
apply (*clarsimp simp add*: *encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def hp-update-score-def*)
by (*metis* (*no-types*, *opaque-lifting*) *hp-child-None-notin hp-child-children-hp-child hp-child-children-simps*(*2*) *hp-child-children-simps*(*3*) *hp-child-hd hp-child-hp-children-simps2 set-mset-union union-iff*)
subgoal
by (*cases x*; *cases h*)
(*auto simp add*: *encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def hp-update-score-def hp-parent-children-cons hp-parent-simps-single-if*)
subgoal
apply (*cases x*; *cases h*)
apply (*clarsimp simp add*: *encoded-hp-prop-list2-conc-def encoded-hp-prop-list-def hp-update-score-def*)
by (*metis hp-node-children-Cons-if hp-node-children-simps2*)
subgoal
by (*cases h*; *cases x*)
(*auto simp*: *hp-update-score-def*)
subgoal
by (*cases h*; *cases x*)
(*auto simp*: *hp-update-score-def*)
subgoal
by (*cases h*; *cases x*)
(*auto simp*: *hp-update-score-def*)
done

lemma *encoded-hp-prop-list-conc-update-score*: ‹*encoded-hp-prop-list-conc arr* ($\mathcal{V}$, *Some* (*Hp a x2 x3*)) ⟹
encoded-hp-prop-list-conc (*hp-update-score′ a* (*Some w′*) *arr*) ($\mathcal{V}$, *Some* (*Hp a w′ x3*))›
supply [*simp*] = *hp-update-score-def*
unfolding *encoded-hp-prop-list-conc-alt-def case-prod-beta encoded-hp-prop-list-def option.case snd-conv*
apply (*intro conjI conjI allI impI ballI*)
subgoal by *auto*
subgoal by (*cases arr*) *auto*
subgoal by (*cases arr*) *auto*

subgoal by (*cases arr*) *auto*
subgoal by (*cases arr*) *auto*
subgoal apply (*cases arr*) **apply** *auto*
by (*metis hp-child-hp-children-simps2*)
subgoal by (*cases arr*) (*auto simp add*: *hp-parent-simps-single-if*)
subgoal by (*cases arr*) *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*cases arr*) *auto*
subgoal by (*cases arr*) *auto*
subgoal by (*cases arr*) *auto*
done

lemma *encoded-hp-prop-list-conc-update-outside*:
‹(*snd h* ≠ *None* ⟹ *a* ∉# *mset-nodes* (*the* (*snd h*))) ⟹ *encoded-hp-prop-list-conc arr h* ⟹
encoded-hp-prop-list-conc (*hp-update-score′ a w′ arr*) *h*›
by (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-list-def*
hp-update-score-def
split: *option.splits*)

definition *ACIDS-decrease-key′* **where**
‹*ACIDS-decrease-key′* = (λ*a w* (𝒱, *h*). (𝒱, *ACIDS.decrease-key a w* (*the h*)))›

lemma *rescale-and-reroot*:
fixes *h* :: ‹*nat multiset* × (*nat*, *nat*)*hp option*›
assumes *enc*: ‹*encoded-hp-prop-list-conc arr h*›
shows ‹*rescale-and-reroot a w′ arr* ≤ ⇓ {(*arr*, *h*). *encoded-hp-prop-list-conc arr h*} (*ACIDS.mop-hm-decrease-key a w′ h*)›
proof −
let *?h* = ‹*snd h*›
have *1*: ‹*encoded-hp-prop-list-conc arr h* ⟹ *encoded-hp-prop-list-conc arr* (*fst h*, *snd h*)›
by (*cases h*) *auto*
have *src*: ‹*source-node arr* = *map-option node ?h*›
using *enc* **by** (*auto simp*: *encoded-hp-prop-list-conc-def split*: *option.splits*)
show *?thesis*
using *assms*
unfolding *rescale-and-reroot-def ACIDS.decrease-key-def ACIDS-decrease-key′-def*
ACIDS.mop-hm-decrease-key-def case-prod-beta[*of - h*] *prod.collapse*
apply (*refine-vcg unroot-hp-tree vsids-merge-pairs*)
subgoal by (*auto simp*: *encoded-hp-prop-list-conc-def split*: *option.splits*)
subgoal by (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def hp-update-score-def split*: *option.splits*)
subgoal by (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def hp-update-score-def split*: *option.splits*)
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr h a*]
apply (*auto split*: *option.splits hp.splits intro*!: *encoded-hp-prop-list-conc-update-outside*)
apply (*metis prod.collapse source-node.simps*)+
done
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr h a*]
in-remove-key-changed[*of a* ‹*the ?h*›] *remove-key-None-iff*[*of a* ‹*the ?h*›] *src*
encoded-hp-prop-list-conc-update-score[*of arr* ‹*fst h*› *a* ‹*score* (*the ?h*)› ‹*hps* (*the ?h*)› *w′*]

apply (*auto split*: *option.splits hp.splits simp*: *find-key-None-remove-key-ident*)
apply (*metis prod.collapse source-node.simps*)+
done
apply (*rule 1*; *assumption*)
subgoal by *auto*
subgoal by *auto*
apply (*rule encoded-hp-prop-list2-conc-update-score*[*of* - ‹*fst h*› ‹*the* (*find-key a* (*the ?h*))› ‹*the* (*remove-key a* (*the ?h*))›])
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr h a*]
in-remove-key-changed[*of a* ‹*the ?h*›] *remove-key-None-iff*[*of a* ‹*the ?h*›]
by (*auto split*: *if-splits simp add*: *find-key-None-remove-key-ident*
encoded-hp-prop-list-conc-def)
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr h a*]
in-remove-key-changed[*of a* ‹*the ?h*›] *remove-key-None-iff*[*of a* ‹*the ?h*›]
find-key-None-or-itself[*of a* ‹*the ?h*›] *find-key-None-remove-key-ident*[*of a* ‹*the ?h*›]
by (*cases* ‹*find-key a* (*the ?h*)›)
(*auto simp del*: *find-key-None-or-itself*)
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr h a*]
in-remove-key-changed[*of a* ‹*the ?h*›] *remove-key-None-iff*[*of a* ‹*the ?h*›]
by (*auto split*: *if-splits simp add*: *find-key-None-remove-key-ident*
encoded-hp-prop-list-conc-def)
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr h a*]
in-remove-key-changed[*of a* ‹*the ?h*›] *remove-key-None-iff*[*of a* ‹*the ?h*›]
node-remove-key-itself-iff[*of a* ‹*the ?h*›]
by (*auto split*: *if-splits simp add*: *find-key-None-remove-key-ident*
encoded-hp-prop-list-conc-def)
subgoal
using *encoded-hp-prop-list-in-node-iff-prev-parent-or-root*[*of arr h a*]
in-remove-key-changed[*of a* ‹*the ?h*›] *remove-key-None-iff*[*of a* ‹*the ?h*›] *src*
find-key-None-or-itself[*of a* ‹*the ?h*›]
by (*cases* ‹*the* (*find-key a* (*the ?h*))›)
(*clarsimp split*: *if-splits simp add*: *find-key-None-remove-key-ident*
simp del: *ACIDS.merge-pairs.simps find-key-None-or-itself*)
done
qed

definition *acids-encoded-hmrel* **where**
‹*acids-encoded-hmrel* = {(*arr*, *h*). *encoded-hp-prop-list-conc arr h*} *O ACIDS.hmrel*›

lemma *hp-insert-spec-mop-prio-insert*:
assumes ‹(*arr*, *h*) ∈ *acids-encoded-hmrel*›
shows ‹*hp-insert i w arr* ≤ ⇓*acids-encoded-hmrel* (*ACIDS.mop-prio-insert i w h*)›
proof −
obtain *j* **where**
i: ‹ *encoded-hp-prop-list-conc arr j*›
‹(*j*, *h*) ∈ *ACIDS.hmrel*›
using *assms* **unfolding** *acids-encoded-hmrel-def* **by** *auto*
show *?thesis*
unfolding *ACIDS.mop-prio-insert-def case-prod-beta acids-encoded-hmrel-def*
apply (*refine-vcg hp-insert-spec*[*THEN order-trans*] *i*)
subgoal using *i* **by** (*auto simp*: *encoded-hp-prop-list-conc-def ACIDS.hmrel-def*)
subgoal using *i* **by** (*auto simp*: *encoded-hp-prop-list-conc-def ACIDS.hmrel-def*)

apply (*rule order-trans*, *rule ref-two-step'*)
apply (*rule ACIDS.mop-prio-insert*)
apply (*rule i*)
apply (*auto simp*: *conc-fun-chain conc-fun-RES ACIDS.mop-prio-insert-def case-prod-beta RETURN-def*)
done
qed

lemma *hp-insert-spec-mop-prio-insert2*:
‹(*uncurry2 hp-insert*, *uncurry2 ACIDS.mop-prio-insert*) ∈
nat-rel $\times_f$ *nat-rel* $\times_f$ *acids-encoded-hmrel* $\to_f$ ⟨*acids-encoded-hmrel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto intro*!: *hp-insert-spec-mop-prio-insert*[*THEN order-trans*])

lemma *rescale-and-reroot-mop-prio-change-weight*:
assumes ‹(*arr*, *h*) ∈ *acids-encoded-hmrel*›
shows ‹*rescale-and-reroot a w arr* ≤ ⇓*acids-encoded-hmrel* (*ACIDS.mop-prio-change-weight a w h*)›
proof −
obtain *j* **where**
i: ‹*encoded-hp-prop-list-conc arr j*›
‹(*j*, *h*) ∈ *ACIDS.hmrel*›
using *assms* **unfolding** *acids-encoded-hmrel-def* **by** *auto*
show *?thesis*
apply (*refine-vcg rescale-and-reroot*[*THEN order-trans*] *i*)
apply (*rule order-trans*, *rule ref-two-step'*)
apply (*rule ACIDS.decrease-key-mop-prio-change-weight i*)+
apply (*auto simp*: *conc-fun-chain conc-fun-RES case-prod-beta RETURN-def acids-encoded-hmrel-def*)
done
qed

lemma *rescale-and-reroot-mop-prio-change-weight2*:
‹(*uncurry2 rescale-and-reroot*, *uncurry2 ACIDS.mop-prio-change-weight*) ∈
nat-rel $\times_f$ *nat-rel* $\times_f$ *acids-encoded-hmrel* $\to_f$ ⟨*acids-encoded-hmrel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto intro*!: *rescale-and-reroot-mop-prio-change-weight*[*THEN order-trans*])

context *hmstruct-with-prio*
begin

definition *mop-hm-is-in* :: ‹-› **where**
‹*mop-hm-is-in w* = (λ(𝒜, *xs*). *do* {
ASSERT (*w* ∈# 𝒜);
RETURN (*xs* ≠ *None* ∧ *w* ∈# *mset-nodes* (*the xs*))
})›

lemma *mop-hm-is-in-mop-prio-is-in*:
assumes ‹(*xs*, *ys*) ∈ *hmrel*›
shows ‹*mop-hm-is-in w xs* ≤ ⇓*bool-rel* (*mop-prio-is-in w ys*)›
using *assms*
unfolding *mop-hm-is-in-def mop-prio-is-in-def*
apply (*refine-vcg*)
subgoal by (*auto simp*: *hmrel-def*)
subgoal by (*auto simp*: *hmrel-def*)
done

lemma *del-min-None-iff*: ‹*del-min* (*Some ya*) = *None* ⟷ *mset-nodes ya* = {#*node ya*#}› **and**
del-min-Some-mset-nodes: ‹*del-min* (*Some ya*) = *Some yb* ⟹ *mset-nodes ya* = *add-mset* (*node ya*) (*mset-nodes yb*)›
apply (*cases ya*; *auto*; *fail*)
apply (*cases ya*; *use mset-nodes-pass*$_2$[*of* ‹*pass*$_1$ (*hps ya*)›] **in** *auto*)
done

lemma *mset-nodes-del-min*[*simp*]:
‹*del-min* (*Some ya*) ≠ *None* ⟹ *mset-nodes* (*the* (*del-min* (*Some ya*))) = *remove1-mset* (*node ya*) (*mset-nodes ya*)›
by (*cases ya*; *auto*)

lemma *hp-score-del-min*:
‹*h* ≠ *None* ⟹ *del-min h* ≠ *None* ⟹ *distinct-mset* (*mset-nodes* (*the h*)) ⟹ *hp-score a* (*the* (*del-min h*)) = (*if a* = *get-min2 h then None else hp-score a* (*the h*))›
using *mset-nodes-pass*$_2$[*of* ‹*pass*$_1$ (*hps* (*the h*))›]
apply (*cases h*; *cases* ‹*the h*›; *cases* ‹*hps* (*the h*) = []›)
apply (*auto simp del*: *mset-nodes-pass*$_2$)
by (*metis hp-score-merge-pairs option.sel option.simps*(*2*) *pairing-heap-assms.pass12-merge-pairs*)

lemma *del-min-prio-del*: ‹(*j*, *h*) ∈ *hmrel* ⟹ *fst* (*snd h*) ≠ {#}⟹
((*fst j*, *del-min* (*snd j*)), *prio-del* (*get-min2* (*snd j*)) *h*) ∈ *hmrel*›
using *hp-score-del-min*[*of* ‹*snd j*›]
apply (*cases* ‹*del-min* (*snd j*)›)
apply (*auto simp*: *hmrel-def ACIDS.prio-del-def del-min-None-iff get-min2-alt-def del-min-Some-mset-nodes*
intro: *invar-del-min dest*: *multi-member-split*)
apply (*metis invar-del-min*)
apply (*metis None-eq-map-option-iff option.map-sel option.sel snd-conv*)
done

definition *mop-hm-old-weight* :: ‹-› **where**
‹*mop-hm-old-weight w* = (λ(𝒜, *xs*). *do* {
ASSERT (*w* ∈# 𝒜);
if xs ≠ *None* ∧ *w* ∈# *mset-nodes* (*the xs*) *then RETURN* (*the* (*hp-score w* (*the xs*)))
else RES UNIV
})›

This requires a stronger invariant than what we want to do.

lemma *mop-hm-old-weight-mop-prio-old-weight*:
‹(*xs*, *ys*) ∈ *hmrel* ⟹ *mop-hm-old-weight w xs* ≤ ⇓*Id* (*mop-prio-old-weight w ys*)›
unfolding *mop-prio-old-weight-def mop-hm-old-weight-def mop-prio-is-in-def nres-monad3*
apply *refine-vcg*
subgoal by (*auto simp*: *hmrel-def*)
subgoal by (*cases* ‹*hp-node w* (*the* (*snd xs*))›)
(*auto simp*: *union-single-eq-member hmrel-def dest*!: *multi-member-split*[*of w*])
done

end

definition *hp-is-in* :: ‹-› **where**
‹*hp-is-in w* = (λ*bw*. *do* {
ASSERT (*w* ∈# *fst bw*);
RETURN (*source-node bw* ≠ *None* ∧ (*hp-read-prev*′ *w bw* ≠ *None* ∨ *hp-read-parent*′ *w bw* ≠ *None* ∨ *the* (*source-node bw*) = *w*))

```
})›

lemma hp-is-in:
  assumes ‹encoded-hp-prop-list-conc arr h›
  shows ‹hp-is-in i arr ≤ ⇓bool-rel (ACIDS.mop-hm-is-in i h)›
proof –
  have dist: ‹source-node arr ≠ None ⟹ distinct-mset (mset-nodes (the (snd h)))›
    ‹source-node arr = None ⟷ snd h = None›
    using assms by (auto simp: encoded-hp-prop-list-conc-def encoded-hp-prop-def
      split: option.splits)
  have rel:
    ‹hp-read-prev′ i arr = map-option node (hp-prev i (the (snd h)))›
    ‹hp-read-parent′ i arr = map-option node (hp-parent i (the (snd h)))›
    if ‹i ∈# fst arr› ‹snd h ≠ None›
    using assms that
    by (cases ‹i ∈# mset-nodes (the (snd h))›;
      auto simp: encoded-hp-prop-list-conc-def encoded-hp-prop-def empty-outside-notin-None
      split: option.splits; fail)+
  show ?thesis
    unfolding hp-is-in-def ACIDS.mop-hm-is-in-def case-prod-beta[of - h]
    apply refine-vcg
    subgoal using assms by (auto simp: encoded-hp-prop-list-conc-def)
    subgoal using rel dist assms in-node-iff-prev-parent-or-root[of ‹the (snd h)› i]
      apply (cases ‹source-node arr = None›)
      apply simp
      apply (simp only:)
      by (smt (verit, best) None-eq-map-option-iff encoded-hp-prop-list-in-node-iff-prev-parent-or-root
        hp-read-fst-snd-simps(2) hp-read-fst-snd-simps(4) option.collapse option.map-sel pair-in-Id-conv
        prod.collapse source-node.simps)
    done
qed

lemma hp-is-in-mop-prio-is-in:
  assumes ‹(arr, h) ∈ acids-encoded-hmrel›
  shows ‹hp-is-in a arr ≤ ⇓bool-rel (ACIDS.mop-prio-is-in a h)›
proof –
  obtain j where
    i: ‹encoded-hp-prop-list-conc arr j›
    ‹(j, h) ∈ ACIDS.hmrel›
    using assms unfolding acids-encoded-hmrel-def by auto
  show ?thesis
    apply (refine-vcg hp-is-in[THEN order-trans] i)
    apply (rule order-trans, rule ref-two-step′)
    apply (rule ACIDS.mop-hm-is-in-mop-prio-is-in i)+
   apply (auto simp: conc-fun-chain conc-fun-RES case-prod-beta RETURN-def acids-encoded-hmrel-def)
    done
qed

lemma hp-is-in-mop-prio-is-in2:
 ‹(uncurry hp-is-in, uncurry ACIDS.mop-prio-is-in) ∈ nat-rel ×f acids-encoded-hmrel →f ⟨bool-rel⟩nres-rel›
  by (intro frefI nres-relI)
   (auto intro!: hp-is-in-mop-prio-is-in[THEN order-trans])

lemma vsids-pop-min2-mop-prio-pop-min:
  fixes arr :: ‹′a::linorder multiset × (′a, nat) hp-fun × ′a option›
  assumes ‹(arr, h) ∈ acids-encoded-hmrel›
```

shows ‹*vsids-pop-min2 arr* ≤ ⇓(*Id*$\times_r$*acids-encoded-hmrel*)(*ACIDS.mop-prio-pop-min h*)›
proof −
obtain *j* **where**
i: ‹*encoded-hp-prop-list-conc arr j*› ‹*encoded-hp-prop-list-conc arr* (*fst j*, *snd j*)›
‹(*j*, *h*) ∈ *ACIDS.hmrel*›
using *assms* **unfolding** *acids-encoded-hmrel-def* **by** *auto*
have *1*: ‹*SPEC*
(λ(*ja*, *arr*).
ja = *get-min2* (*snd j*) ∧
encoded-hp-prop-list-conc arr (*fst j*, *ACIDS.del-min* (*snd j*)))
≤ ⇓ (*Id* $\times_f$ *acids-encoded-hmrel*)
(*do* {
v ← *SPEC* (*ACIDS.prio-peek-min* (*fst h*, *fst* (*snd h*), *snd* (*snd h*)));
x ← *ASSERT* (*v* ∈# *fst* (*snd h*) ∧ *v* ∈# *fst h*);
bw ← *RETURN* (*ACIDS.prio-del v* (*fst h*, *fst* (*snd h*), *snd* (*snd h*)));
RETURN (*v*, *bw*)
})› (**is** ‹*?A* ≤ ⇓ - *?B*›)
if ‹*fst* (*snd h*) ≠ {#}›
proof −
have *A*: ‹*?A* = *do* {
let ja = *get-min2* (*snd j*);
bw ← *SPEC* (λ*bw*. *encoded-hp-prop-list-conc bw* (*fst j*, *ACIDS.del-min* (*snd j*)));
RETURN (*ja*, *bw*)
}›
by (*auto simp*: *RETURN-def conc-fun-RES RES-RES-RETURN-RES*)
have *1*: ‹(*get-min2* (*snd j*), *v*) ∈ *Id* ⟹ *v* ∈# *fst* (*snd h*) ∧ *v* ∈# *fst h* ⟹
encoded-hp-prop-list-conc x (*fst j*, *ACIDS.del-min* (*snd j*)) ⟹
(*x*, *ACIDS.prio-del v* (*fst h*, *fst* (*snd h*), *snd* (*snd h*))) ∈ *acids-encoded-hmrel*›
for *v x*
using *ACIDS.del-min-prio-del*[*of j h*] *i that*
by (*auto simp*: *acids-encoded-hmrel-def*)
show *?thesis*
unfolding *A*
apply *refine-vcg*
subgoal using *i that* **apply** (*cases* ‹*the* (*snd j*)›) **apply** (*auto simp*: *ACIDS.prio-peek-min-def*
ACIDS.hmrel-def ACIDS.invar-def in-mset-sum-list-iff
encoded-hp-prop-list-conc-def
ACIDS.set-hp-is-hp-score-mset-nodes)
apply (*drule bspec*, *assumption*)
apply (*subst* (*asm*) *ACIDS.set-hp-is-hp-score-mset-nodes*)
apply (*auto simp*: *encoded-hp-prop-def distinct-mset-add dest*!: *split-list multi-member-split*)
by (*metis hp-node-None-notin2 member-add-mset option.map-sel*)
apply (*rule 1*; *assumption*)
subgoal by *auto*
done
qed

show *?thesis*
unfolding *ACIDS.mop-prio-pop-min-def ACIDS.mop-prio-peek-min-def*
ACIDS.mop-prio-del-def nres-monad2 case-prod-beta[*of* - *h*] *case-prod-beta*[*of* - ‹*snd h*›] *nres-monad3*
apply (*refine-vcg vsids-pop-min2*[*THEN order-trans*] *i 1*)
subgoal using *i* **by** (*auto simp*: *ACIDS.hmrel-def*)
done
qed

lemma *vsids-pop-min2-mop-prio-pop-min2*:

‹(*vsids-pop-min2*, *ACIDS.mop-prio-pop-min*) ∈ *acids-encoded-hmrel* $\rightarrow_f$ ⟨*nat-rel* $\times_r$ *acids-encoded-hmrel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto intro*!: *vsids-pop-min2-mop-prio-pop-min*[*THEN order-trans*])

definition *mop-hp-read-score* :: ‹-› **where**
‹*mop-hp-read-score x* = (λ(𝒜, *w*, *h*). *do* {
ASSERT (*x* ∈# 𝒜);
if hp-read-score x w ≠ *None then RETURN* (*the* (*hp-read-score x w*)) *else RES UNIV*
}) ›

lemma *mop-hp-read-score-mop-hm-old-weight*:
assumes ‹*encoded-hp-prop-list-conc arr h*›
shows
‹*mop-hp-read-score w arr* ≤ ⇓*Id* (*ACIDS.mop-hm-old-weight w h*)›
proof −
show *?thesis*
unfolding *mop-hp-read-score-def ACIDS.mop-hm-old-weight-def RETURN-def RES-RES-RETURN-RES Many-More.if-f*
apply *refine-vcg*
subgoal using *assms* **by** (*auto simp*: *encoded-hp-prop-list-conc-def*)
subgoal using *assms* **by** (*auto simp*: *encoded-hp-prop-list-conc-def encoded-hp-prop-def split*: *option.splits*)
done
qed

lemma *mop-hp-read-score-mop-prio-old-weight*:
fixes *arr* :: ‹*'a*::*linorder multiset* × (*'a*, *nat*) *hp-fun* × *'a option*›
assumes ‹(*arr*, *h*) ∈ *acids-encoded-hmrel*›
shows ‹*mop-hp-read-score w arr* ≤ ⇓(*Id*)(*ACIDS.mop-prio-old-weight w h*)›
proof −
obtain *j* **where**
i: ‹*encoded-hp-prop-list-conc arr j*› ‹*encoded-hp-prop-list-conc arr* (*fst j*, *snd j*)›
‹(*j*, *h*) ∈ *ACIDS.hmrel*›
using *assms* **unfolding** *acids-encoded-hmrel-def* **by** *auto*
show *?thesis*
apply (*rule mop-hp-read-score-mop-hm-old-weight*[*THEN order-trans*] *i*)+
subgoal
by (*rule ref-two-step' ACIDS.mop-hm-old-weight-mop-prio-old-weight*[*THEN order-trans*] *i*)+
auto
done
qed

lemma *mop-hp-read-score-mop-prio-old-weight2*:
‹(*uncurry mop-hp-read-score*, *uncurry ACIDS.mop-prio-old-weight*) ∈ *nat-rel* $\times_r$ *acids-encoded-hmrel* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto intro*!: *mop-hp-read-score-mop-prio-old-weight*[*THEN order-trans*])

thm *ACIDS.mop-prio-insert-raw-unchanged-def*
thm *ACIDS.mop-prio-insert-maybe-def*
term *ACIDS.prio-peek-min*
thm *ACIDS.mop-prio-old-weight-def*
thm *ACIDS.mop-prio-insert-raw-unchanged-def*
term *ACIDS.mop-prio-insert-unchanged*
end

theory *Pairing-Heaps-Impl*
  imports *Relational-Pairing-Heaps*
    Map-Fun-Rel
begin

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN NEMonad.SPEC*

1.2 Imperative Pairing heaps

type-synonym ($'a,'b$)*pairing-heaps-imp* = ‹($'a$ *option list* × $'a$ *option list* × $'a$ *option list* × $'a$ *option list* × $'b$ *list* × $'a$ *option*)›
definition *pairing-heaps-rel* :: ‹($'a$ *option* × *nat option*) *set* ⇒ ($'b$ *option* ×$'c$ *option*) *set* ⇒
  (($'a,'b$)*pairing-heaps-imp* × (*nat multiset* × (*nat*,$'c$) *hp-fun* × *nat option*)) *set*› **where**
pairing-heaps-rel-def-internal:
  ‹*pairing-heaps-rel R S* = {((*prevs'*, *nxts'*, *children'*, *parents'*, *scores'*, *h'*), ($\mathcal{V}$, (*prevs*, *nxts*, *children*, *parents*, *scores*), *h*)).
    (*h'*, *h*) ∈ *R* ∧
    (*prevs'*, *prevs*) ∈ ⟨*R*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$) ∧
    (*nxts'*, *nxts*) ∈ ⟨*R*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$) ∧
    (*children'*, *children*) ∈ ⟨*R*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$) ∧
    (*parents'*, *parents*) ∈ ⟨*R*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$) ∧
    (*map Some scores'*, *scores*) ∈ ⟨*S*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$)
  }›

lemma *pairing-heaps-rel-def*:
  ‹⟨*R*,*S*⟩*pairing-heaps-rel* =
{((*prevs'*, *nxts'*, *children'*, *parents'*, *scores'*, *h'*), ($\mathcal{V}$, (*prevs*, *nxts*, *children*, *parents*, *scores*), *h*)).
    (*h'*, *h*) ∈ *R* ∧
    (*prevs'*, *prevs*) ∈ ⟨*R*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$) ∧
    (*nxts'*, *nxts*) ∈ ⟨*R*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$) ∧
    (*children'*, *children*) ∈ ⟨*R*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$) ∧
    (*parents'*, *parents*) ∈ ⟨*R*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$) ∧
    (*map Some scores'*, *scores*) ∈ ⟨*S*⟩*map-fun-rel* ((λ*a*. (*a*,*a*))‘ *set-mset* $\mathcal{V}$)
  }›
  unfolding *pairing-heaps-rel-def-internal relAPP-def* **by** *auto*

definition *op-hp-read-nxt-imp* **where**
  ‹*op-hp-read-nxt-imp* = (λ*i* (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
      (*nxts* ! *i*)
  })›
definition *mop-hp-read-nxt-imp* **where**
  ‹*mop-hp-read-nxt-imp* = (λ*i* (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
      ASSERT (*i* < *length nxts*);
      RETURN (*nxts* ! *i*)
  })›

lemma *op-hp-read-nxt-imp-spec*:
  ‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈*nat-rel* ⟹ *j* ∈# *fst ys* ⟹
  (*op-hp-read-nxt-imp i xs*, *hp-read-nxt' j ys*) ∈ *R*›
  unfolding *op-hp-read-nxt-imp-def*
  by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)

lemma *mop-hp-read-nxt-imp-spec*:
  ‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈*nat-rel* ⟹ *j* ∈# *fst ys* ⟹

mop-hp-read-nxt-imp i xs ≤ *SPEC* (λ*a*. (*a*, *hp-read-nxt′ j ys*) ∈ *R*)›
unfolding *mop-hp-read-nxt-imp-def*
apply *refine-vcg*
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
done

definition *op-hp-read-prev-imp* **where**
‹*op-hp-read-prev-imp* = (λ*i* (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
prevs ! *i*
})›

definition *mop-hp-read-prev-imp* **where**
‹*mop-hp-read-prev-imp* = (λ*i* (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
ASSERT (*i* < *length prevs*);
RETURN (*prevs* ! *i*)
})›

lemma *op-hp-read-prev-imp-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈*nat-rel* ⟹ *j* ∈# *fst ys* ⟹
(*op-hp-read-prev-imp i xs*, *hp-read-prev′ j ys*) ∈ *R*›
unfolding *op-hp-read-prev-imp-def*
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)

lemma *mop-hp-read-prev-imp-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈*nat-rel* ⟹ *j* ∈# *fst ys* ⟹
mop-hp-read-prev-imp i xs ≤ *SPEC* (λ*a*. (*a*, *hp-read-prev′ j ys*) ∈ *R*)›
unfolding *mop-hp-read-prev-imp-def*
apply *refine-vcg*
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
done

definition *op-hp-read-child-imp* **where**
‹*op-hp-read-child-imp* = (λ*i* (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
(*children* ! *i*)
})›

definition *mop-hp-read-child-imp* **where**
‹*mop-hp-read-child-imp* = (λ*i* (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
ASSERT (*i* < *length children*);
RETURN (*children* ! *i*)
})›

lemma *op-hp-read-child-imp-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈*nat-rel* ⟹ *j* ∈# *fst ys* ⟹
(*op-hp-read-child-imp i xs*, *hp-read-child′ j ys*) ∈ *R*›
unfolding *op-hp-read-child-imp-def*
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)

lemma *mop-hp-read-child-imp-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈*nat-rel* ⟹ *j* ∈# *fst ys* ⟹

mop-hp-read-child-imp i xs $\leq$ *SPEC* (λa. (*a*, *hp-read-child' j ys*) $\in$ *R*)›
unfolding *mop-hp-read-child-imp-def*
apply *refine-vcg*
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
done

definition *mop-hp-read-parent-imp* **where**
‹*mop-hp-read-parent-imp* = (λi (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
ASSERT (*i* < *length parents*);
RETURN (*parents* ! *i*)
})›
definition *op-hp-read-parent-imp* **where**
‹*op-hp-read-parent-imp* = (λi (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
(*parents* ! *i*)
})›

lemma *op-hp-read-parent-imp-spec*:
‹(*xs*, *ys*) $\in$ $\langle R,S\rangle$*pairing-heaps-rel* $\Longrightarrow$ (*i*,*j*)$\in$*nat-rel* $\Longrightarrow$ *j* $\in$# *fst ys* $\Longrightarrow$
(*op-hp-read-parent-imp i xs*, *hp-read-parent' j ys*) $\in$ *R*›
unfolding *op-hp-read-parent-imp-def*
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)

lemma *mop-hp-read-parent-imp-spec*:
‹(*xs*, *ys*) $\in$ $\langle R,S\rangle$*pairing-heaps-rel* $\Longrightarrow$ (*i*,*j*)$\in$*nat-rel* $\Longrightarrow$ *j* $\in$# *fst ys* $\Longrightarrow$
mop-hp-read-parent-imp i xs $\leq$ *SPEC* (λa. (*a*, *hp-read-parent' j ys*) $\in$ *R*)›
unfolding *mop-hp-read-parent-imp-def*
apply *refine-vcg*
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
done

definition *op-hp-read-score-imp* :: ‹*nat* ⇒ (*'a*,*'b*)*pairing-heaps-imp* ⇒ *'b*› **where**
‹*op-hp-read-score-imp* = (λi (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
((*scores* ! *i*))
})›

definition *mop-hp-read-score-imp* :: ‹*nat* ⇒ (*'a*,*'b*)*pairing-heaps-imp* ⇒ *'b nres*› **where**
‹*mop-hp-read-score-imp* = (λi (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
ASSERT (*i* < *length scores*);
RETURN ((*scores* ! *i*))
})›

lemma *mop-hp-read-score-imp-spec*:
‹(*xs*, *ys*) $\in$ $\langle R,S\rangle$*pairing-heaps-rel* $\Longrightarrow$ (*i*,*j*)$\in$*nat-rel* $\Longrightarrow$ *j* $\in$# *fst ys* $\Longrightarrow$
mop-hp-read-score-imp i xs $\leq$ *SPEC* (λa. (*Some a*, *hp-read-score' j ys*) $\in$ *S*)›
unfolding *mop-hp-read-score-imp-def*
apply *refine-vcg*
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def dest*!: *bspec*)

done

fun *hp-set-all′* **where**
‹*hp-set-all′ i p q r s t* ($\mathcal{V}$, *u*, *h*) = ($\mathcal{V}$, *hp-set-all i p q r s t u*, *h*)›

definition *mop-hp-set-all-imp* :: ‹*nat* ⇒ - ⇒ - ⇒ - ⇒ - ⇒ - ⇒ (′*a*,′*b*)*pairing-heaps-imp* ⇒ (′*a*,′*b*)*pairing-heaps-imp nres*›**where**
‹*mop-hp-set-all-imp* = (λ*i p q r s t* (*prevs*, *nxts*, *children*, *parents*, *scores*, *h*). *do* {
ASSERT (*i* < *length nxts* ∧ *i* < *length prevs* ∧ *i* < *length parents* ∧ *i* < *length children* ∧ *i* < *length scores*);
RETURN (*prevs*[*i* := *p*], *nxts*[*i*:=*q*], *children*[*i*:=*r*], *parents*[*i*:=*s*], *scores*[*i*:=*t*], *h*)
})›

lemma *mop-hp-set-all-imp-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹(*i*,*j*)∈*nat-rel* ⟹
(*p*′,*p*)∈*R* ⟹ (*q*′,*q*)∈*R*⟹(*r*′,*r*)∈*R* ⟹ (*s*′,*s*)∈*R*⟹ (*Some t*′,*t*)∈*S* ⟹ *j* ∈# *fst ys* ⟹
mop-hp-set-all-imp i p′ *q*′ *r*′ *s*′ *t*′ *xs* ≤ *SPEC* (λ*a*. (*a*, *hp-set-all*′ *j p q r s t ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel*)›
unfolding *mop-hp-set-all-imp-def*
apply *refine-vcg*
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)
subgoal
by (*force simp*: *pairing-heaps-rel-def map-fun-rel-def hp-set-all-def*)
done

lemma *fst-hp-set-all*′[*simp*]: ‹*fst* (*hp-set-all*′ *i p q r s t x*) = *fst x*›
by (*cases x*) *auto*

fun *update-source-node-impl* :: ‹- ⇒ (′*a*,′*b*)*pairing-heaps-imp* ⇒ (′*a*,′*b*)*pairing-heaps-imp*› **where**
‹*update-source-node-impl i* (*prevs*, *nxts*, *parents*, *children*, *scores*,-) = (*prevs*, *nxts*, *parents*, *children*, *scores*, *i*)›

fun *source-node-impl* :: ‹(′*a*,′*b*)*pairing-heaps-imp* ⇒ ′*a option*› **where**
‹*source-node-impl* (*prevs*, *nxts*, *parents*, *children*, *scores*,*h*) = *h*›

lemma *update-source-node-impl-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*) ∈ *R* ⟹
(*update-source-node-impl i xs*, *update-source-node j ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel*›
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)

lemma *source-node-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹
(*source-node-impl xs*, *source-node ys*) ∈ *R*›
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def*)

lemma *hp-insert-alt-def*:
‹*hp-insert* = (λ*i w arr*. *do* {

```
  let h = source-node arr;
  if h = None then do {
    ASSERT (i ∈# fst arr);
    let arr = (hp-set-all' i None None None None (Some w) arr);
    RETURN (update-source-node (Some i) arr)
   } else do {
    ASSERT (i ∈# fst arr);
    ASSERT (hp-read-prev' i arr = None);
    ASSERT (hp-read-parent' i arr = None);
    let j = the h;
    ASSERT (j ∈# (fst arr) ∧ j ≠ i);
    ASSERT (hp-read-score' j (arr) ≠ None);
    ASSERT (hp-read-prev' j arr = None ∧ hp-read-nxt' j arr = None ∧ hp-read-parent' j arr = None);
    let y = (the (hp-read-score' j arr));
    if y < w
    then do {
      let arr = hp-set-all' i None None (Some j) None (Some w) arr;
      ASSERT (j ∈# fst arr);
      let arr = hp-update-parents' j (Some i) arr;
      RETURN (update-source-node (Some i) arr)
    }
    else do {
      let child = hp-read-child' j arr;
      ASSERT (child ≠ None ⟶ the child ∈# fst arr);
      let arr = hp-set-all' j None None (Some i) None (Some y) arr;
      ASSERT (i ∈# fst arr);
      let arr = hp-set-all' i None child None (Some j) (Some (w)) arr;
      ASSERT (child ≠ None ⟶ the child ∈# fst arr);
      let arr = (if child = None then arr else hp-update-prev' (the child) (Some i) arr);
      ASSERT (child ≠ None ⟶ the child ∈# fst arr);
      let arr = (if child = None then arr else hp-update-parents' (the child) None arr);
      RETURN arr
    }
   }
      })› (is ‹?A = ?B›)
proof -
  have ‹?A i w arr ≤ ⇓Id (?B i w arr)› for i w arr
    unfolding hp-insert-def
    by refine-vcg (solves ‹auto intro!: simp: Let-def›)+
  moreover have ‹?B i w arr ≤ ⇓Id (?A i w arr)› for i w arr
    unfolding hp-insert-def
    by refine-vcg (auto intro!: ext bind-cong[OF refl] simp: Let-def)
  ultimately show ?thesis unfolding Down-id-eq apply -
    apply (intro ext)
    apply (rule antisym)
    apply assumption+
    done
qed

definition mop-hp-update-prev'-imp :: ‹nat ⇒ 'a option ⇒ ('a,'b)pairing-heaps-imp ⇒ ('a,'b)pairing-heaps-imp
nres› where
  ‹mop-hp-update-prev'-imp = (λi v (prevs, nxts, parents, children). do {
    ASSERT (i < length prevs);
    RETURN (prevs[i:=v], nxts, parents, children)
  })›
```

lemma *mop-hp-update-prev′-imp-spec*:
‹$(xs, ys) \in \langle R,S\rangle pairing\text{-}heaps\text{-}rel \Longrightarrow j \in\# fst\ ys \Longrightarrow (i,j)\in nat\text{-}rel \Longrightarrow$
$(p',p)\in R \Longrightarrow$
$mop\text{-}hp\text{-}update\text{-}prev'\text{-}imp\ i\ p'\ xs \leq SPEC\ (\lambda a.\ (a,\ hp\text{-}update\text{-}prev'\ j\ p\ ys) \in \langle R,S\rangle pairing\text{-}heaps\text{-}rel)$›
unfolding *mop-hp-update-prev′-imp-def*
apply *refine-vcg*
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def hp-update-prev-def*)
subgoal
by (*force simp*: *pairing-heaps-rel-def map-fun-rel-def hp-update-prev-def*)
done

definition *mop-hp-update-parent′-imp* :: ‹$nat \Rightarrow 'a\ option \Rightarrow ('a,'b)pairing\text{-}heaps\text{-}imp \Rightarrow ('a,'b)pairing\text{-}heaps\text{-}imp$ $nres$› **where**
‹$mop\text{-}hp\text{-}update\text{-}parent'\text{-}imp = (\lambda i\ v\ (prevs,\ nxts,children,\ parents,\ scores).\ do\ \{$
$ASSERT\ (i < length\ parents);$
$RETURN\ (prevs,\ nxts,\ children,\ parents[i:=v],\ scores)$
$\})$›

lemma *mop-hp-update-parent′-imp-spec*:
‹$(xs, ys) \in \langle R,S\rangle pairing\text{-}heaps\text{-}rel \Longrightarrow j \in\# fst\ ys \Longrightarrow (i,j)\in nat\text{-}rel \Longrightarrow$
$(p',p)\in R \Longrightarrow$
$mop\text{-}hp\text{-}update\text{-}parent'\text{-}imp\ i\ p'\ xs \leq SPEC\ (\lambda a.\ (a,\ hp\text{-}update\text{-}parents'\ j\ p\ ys) \in \langle R,S\rangle pairing\text{-}heaps\text{-}rel)$›
unfolding *mop-hp-update-parent′-imp-def*
apply *refine-vcg*
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def hp-update-parents-def*)
subgoal
by (*force simp*: *pairing-heaps-rel-def map-fun-rel-def hp-update-parents-def*)
done

definition *mop-hp-update-nxt′-imp* :: ‹$nat \Rightarrow 'a\ option \Rightarrow ('a,'b)pairing\text{-}heaps\text{-}imp \Rightarrow ('a,'b)pairing\text{-}heaps\text{-}imp$ $nres$› **where**
‹$mop\text{-}hp\text{-}update\text{-}nxt'\text{-}imp = (\lambda i\ v\ (prevs,\ nxts,\ parents,\ children).\ do\ \{$
$ASSERT\ (i < length\ nxts);$
$RETURN\ (prevs,\ nxts[i:=v],\ parents,\ children)$
$\})$›

lemma *mop-hp-update-nxt′-imp-spec*:
‹$(xs, ys) \in \langle R,S\rangle pairing\text{-}heaps\text{-}rel \Longrightarrow j \in\# fst\ ys \Longrightarrow (i,j)\in nat\text{-}rel \Longrightarrow$
$(p',p)\in R \Longrightarrow$
$mop\text{-}hp\text{-}update\text{-}nxt'\text{-}imp\ i\ p'\ xs \leq SPEC\ (\lambda a.\ (a,\ hp\text{-}update\text{-}nxt'\ j\ p\ ys) \in \langle R,S\rangle pairing\text{-}heaps\text{-}rel)$›
unfolding *mop-hp-update-nxt′-imp-def*
apply *refine-vcg*
subgoal
by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def hp-update-prev-def*)
subgoal
by (*force simp*: *pairing-heaps-rel-def map-fun-rel-def hp-update-nxt-def*)
done

definition *mop-hp-update-score-imp* :: ‹$nat \Rightarrow 'b \Rightarrow ('a,'b)pairing\text{-}heaps\text{-}imp \Rightarrow ('a,'b)pairing\text{-}heaps\text{-}imp$ $nres$› **where**

‹mop-hp-update-score-imp = (λi v (prevs, nxts, parents, children, scores, h). do {
  ASSERT (i < length scores);
  RETURN (prevs, nxts, parents, children, scores[i:=v], h)
})›

lemma *mop-hp-update-score-imp-spec*:
  ‹(xs, ys) ∈ ⟨R,S⟩pairing-heaps-rel ⟹ (i,j)∈nat-rel ⟹ j ∈# fst ys ⟹
   (Some p′, p)∈S ⟹
  mop-hp-update-score-imp i p′ xs ≤ SPEC (λa. (a, hp-update-score′ j p ys) ∈ ⟨R,S⟩pairing-heaps-rel)›
  unfolding *mop-hp-update-score-imp-def*
  apply *refine-vcg*
  subgoal
    by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def hp-update-prev-def*)
  subgoal
    by (*force simp*: *pairing-heaps-rel-def map-fun-rel-def hp-update-score-def*)
  done

definition *mop-hp-update-child′-imp* :: ‹nat ⇒ ′a option ⇒ (′a,′b)pairing-heaps-imp ⇒ (′a,′b)pairing-heaps-imp nres› **where**
  ‹mop-hp-update-child′-imp = (λi v (prevs, nxts, children, parents, scores). do {
    ASSERT (i < length children);
    RETURN (prevs, nxts, children[i:=v], parents, scores)
  })›

lemma *mop-hp-update-child′-imp-spec*:
  ‹(xs, ys) ∈ ⟨R,S⟩pairing-heaps-rel ⟹ j ∈# fst ys ⟹ (i,j)∈nat-rel ⟹
   (p′,p)∈R ⟹
  mop-hp-update-child′-imp i p′ xs ≤ SPEC (λa. (a, hp-update-child′ j p ys) ∈ ⟨R,S⟩pairing-heaps-rel)›
  unfolding *mop-hp-update-child′-imp-def*
  apply *refine-vcg*
  subgoal
    by (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def hp-update-prev-def*)
  subgoal
    by (*force simp add*: *pairing-heaps-rel-def map-fun-rel-def hp-update-child-def*)
  done

definition *mop-hp-insert-impl* :: ‹nat ⇒ ′b::linorder ⇒ (nat,′b)pairing-heaps-imp ⇒ (nat,′b)pairing-heaps-imp nres› **where**
  ‹mop-hp-insert-impl = (λi (w::′b) (arr :: (nat,′b)pairing-heaps-imp). do {
  let h = source-node-impl arr;
  if h = None then do {
    arr ← mop-hp-set-all-imp i None None None None w arr;
    RETURN (update-source-node-impl (Some i) arr)
   } else do {
    ASSERT (op-hp-read-prev-imp i arr = None);
    ASSERT (op-hp-read-parent-imp i arr = None);
    let j = (the h);
   ASSERT (op-hp-read-prev-imp j arr = None ∧ op-hp-read-nxt-imp j arr = None ∧ op-hp-read-parent-imp j arr = None);
    y ← mop-hp-read-score-imp j arr;
    if y < w
    then do {

arr ← mop-hp-set-all-imp i None None (Some j) None ((w)) (arr);
arr← mop-hp-update-parent′-imp j (Some i) arr;
RETURN (update-source-node-impl (Some i) arr)
}
else do {
child ← mop-hp-read-child-imp j arr;
arr ← mop-hp-set-all-imp j None None (Some i) None (y) arr;
arr ← mop-hp-set-all-imp i None child None (Some j) w arr;
arr ← (if child = None then RETURN arr else mop-hp-update-prev′-imp (the child) (Some i) arr);
arr ← (if child = None then RETURN arr else mop-hp-update-parent′-imp (the child) None arr);
RETURN arr
}
}
})›

lemma *Some-x-y-option-theD*: ‹(Some x, y) ∈ ⟨S⟩option-rel ⟹ (x, the y) ∈ S›
by (*auto simp*: *option-rel-def*)

context
begin
private lemma *in-pairing-heaps-rel-still*: ‹(arra, arr′) ∈ ⟨⟨nat-rel⟩option-rel, ⟨S⟩option-rel⟩pairing-heaps-rel ⟹ arr′ = arr′′ ⟹
(arra, arr′′) ∈ ⟨⟨nat-rel⟩option-rel, ⟨S⟩option-rel⟩pairing-heaps-rel›
by *auto*

lemma *mop-hp-insert-impl-spec*:
assumes ‹(xs, ys) ∈ ⟨⟨nat-rel⟩option-rel,⟨nat-rel⟩option-rel⟩pairing-heaps-rel› ‹(i,j)∈nat-rel› ‹(w,w′)∈nat-rel›
shows ‹mop-hp-insert-impl i w xs ≤ ⇓(⟨⟨nat-rel⟩option-rel,⟨nat-rel⟩option-rel⟩pairing-heaps-rel) (hp-insert j w′ ys)›
proof −
have [*refine*]: ‹(Some i, Some j) ∈ ⟨nat-rel⟩option-rel›
using *assms* **by** *auto*
have *K*: ‹hp-read-child′ (the (source-node ys)) ys ≠ None ⟶
the (hp-read-child′ (the (source-node ys)) ys) ∈ 𝒱 ⟹ the (source-node ys) ∈# fst ys ⟹
op-hp-read-child-imp (the (source-node-impl xs)) xs ≠ None ⟹
the (op-hp-read-child-imp (the (source-node ys)) xs) ∈ 𝒱› **for** 𝒱
using *op-hp-read-child-imp-spec*[*of xs ys* ‹⟨nat-rel⟩option-rel› ‹⟨nat-rel⟩option-rel› ‹the (source-node ys)› ‹the (source-node-impl xs)›]
source-node-spec[*of xs ys* ‹⟨nat-rel⟩option-rel› ‹⟨nat-rel⟩option-rel›] *assms*
by *auto*
show *?thesis*
using *assms*
unfolding *mop-hp-insert-impl-def hp-insert-alt-def*
apply (*refine-vcg mop-hp-set-all-imp-spec*[**where** R=‹⟨nat-rel⟩option-rel› **and** S=‹⟨nat-rel⟩option-rel›]
mop-hp-read-score-imp-spec[**where** R=‹⟨nat-rel⟩option-rel› **and** S=‹⟨nat-rel⟩option-rel› **and** ys=ys **and** j=‹the (source-node-impl xs)›]
Some-x-y-option-theD[**where** S=nat-rel]
mop-hp-update-parent′-imp-spec[**where** R=‹⟨nat-rel⟩option-rel› **and** S=‹⟨nat-rel⟩option-rel›]
mop-hp-read-child-imp-spec[**where** R=‹⟨nat-rel⟩option-rel› **and** S=‹⟨nat-rel⟩option-rel›]
mop-hp-update-prev′-imp-spec[**where** R=‹⟨nat-rel⟩option-rel› **and** S=‹⟨nat-rel⟩option-rel› **and** j=‹the (hp-read-child′ (the (source-node ys)) ys)›])
subgoal by (*auto dest*: *source-node-spec*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *update-source-node-impl-spec simp*: *refl-on-def*)
subgoal by (*auto dest*!: *op-hp-read-prev-imp-spec*)
subgoal by (*auto dest*!: *op-hp-read-parent-imp-spec*)
subgoal
using *op-hp-read-parent-imp-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*› ‹*the* (*source-node ys*)› ‹*the* (*source-node-impl xs*)›]
source-node-spec[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
apply *auto*
by (*metis op-hp-read-prev-imp-spec pair-in-Id-conv*)
subgoal
using *op-hp-read-nxt-imp-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*› ‹*the* (*source-node ys*)› ‹*the* (*source-node-impl xs*)›]
source-node-spec[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
by *auto*
subgoal
using *op-hp-read-parent-imp-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*› ‹*the* (*source-node ys*)› ‹*the* (*source-node-impl xs*)›]
source-node-spec[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
by *auto*
subgoal by *auto*
subgoal
using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
by *auto*
subgoal
using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›] **by** *auto*
subgoal by (*auto intro*!: *update-source-node-impl-spec*)
subgoal using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›] **by** *auto*
subgoal by (*auto intro*!: *update-source-node-impl-spec*)
subgoal using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›] **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal *HH*
using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
op-hp-read-child-imp-spec[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*› ‹*the* (*source-node ys*)› ‹*the* (*source-node-impl xs*)›]
by *auto*
subgoal
using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›] **by** *auto*
subgoal
using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›] **by** *auto*
subgoal using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›] **by** *auto*
subgoal by *auto*

apply (*rule in-pairing-heaps-rel-still*, *assumption*)
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal
using *op-hp-read-child-imp-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
by (*metis HH option.collapse*)
subgoal
using *HH* **by** *auto*
apply (*rule in-pairing-heaps-rel-still*, *assumption*)
subgoal
by *auto*
apply (*assumption*)
apply (*rule K*)
apply *assumption*
subgoal by *auto*
subgoal
using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
op-hp-read-child-imp-spec[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*› ‹*the* (*source-node ys*)›
‹*the* (*source-node-impl xs*)›]
by *auto*
subgoal
apply (*frule K*)
by *auto*
(*metis BNF-Greatest-Fixpoint.IdD assms*(*1*) *op-hp-read-child-imp-spec option-rel-simp*(*3*) *source-node-spec*)+
apply (*rule autoref-opt*(*1*))
subgoal
using *source-node-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›]
op-hp-read-child-imp-spec[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*› ‹*the* (*source-node ys*)›
‹*the* (*source-node-impl xs*)›]
by *auto*
subgoal by *auto*
done
qed

lemma *hp-link-alt-def*:
‹*hp-link* = (λ(*i*::′*a*) *j arr*. *do* {
ASSERT ($i \neq j$);
ASSERT (*i* ∈# *fst arr*);
ASSERT (*j* ∈# *fst arr*);
ASSERT (*hp-read-score′ i arr* ≠ *None*);
ASSERT (*hp-read-score′ j arr* ≠ *None*);
let x = (*the* (*hp-read-score′ i arr*)::′*b*::*order*);
let y = (*the* (*hp-read-score′ j arr*)::′*b*);
let prev = *hp-read-prev′ i arr*;
let nxt = *hp-read-nxt′ j arr*;
ASSERT (*nxt* ≠ *Some i* ∧ *nxt* ≠ *Some j*);
ASSERT (*prev* ≠ *Some i* ∧ *prev* ≠ *Some j*);
let (*parent*,*ch*,w_p, w_{ch}) = (*if* $y < x$ *then* (*i*, *j*, *x*, *y*) *else* (*j*, *i*, *y*, *x*));
ASSERT (*parent* ∈# *fst arr*);
ASSERT (*ch* ∈# *fst arr*);
let child = *hp-read-child′ parent arr*;
ASSERT (*child* ≠ *Some i* ∧ *child* ≠ *Some j*);
let $child_{ch}$ = *hp-read-child′ ch arr*;
ASSERT ($child_{ch}$ ≠ *Some i* ∧ $child_{ch}$ ≠ *Some j* ∧ ($child_{ch}$ ≠ *None* ⟶ $child_{ch}$ ≠ *child*));
ASSERT (*distinct* ([*i*, *j*] @ (*if* $child_{ch}$ ≠ *None then* [*the* $child_{ch}$] *else* [])

@ (*if child* $\neq$ *None then* [*the child*] *else* [])
@ (*if prev* $\neq$ *None then* [*the prev*] *else* [])
@ (*if nxt* $\neq$ *None then* [*the nxt*] *else* []))
);
ASSERT (*ch* $\in\#$ *fst arr*);
ASSERT (*parent* $\in\#$ *fst arr*);
ASSERT (*child* $\neq$ *None* $\longrightarrow$ *the child* $\in\#$ *fst arr*);
ASSERT (*nxt* $\neq$ *None* $\longrightarrow$ *the nxt* $\in\#$ *fst arr*);
ASSERT (*prev* $\neq$ *None* $\longrightarrow$ *the prev* $\in\#$ *fst arr*);
let arr = *hp-set-all′ parent prev nxt* (*Some ch*) *None* (*Some* (w_p::′*b*)) *arr*;
let arr = *hp-set-all′ ch None child* $child_{ch}$ (*Some parent*) (*Some* (w_{ch}::′*b*)) *arr*;
let arr = (*if child* = *None then arr else hp-update-prev′* (*the child*) (*Some ch*) *arr*);
let arr = (*if nxt* = *None then arr else hp-update-prev′* (*the nxt*) (*Some parent*) *arr*);
let arr = (*if prev* = *None then arr else hp-update-nxt′* (*the prev*) (*Some parent*) *arr*);
let arr = (*if child* = *None then arr else hp-update-parents′* (*the child*) *None arr*);
RETURN (*arr*, *parent*)
})› (**is** ‹*?A* = *?B*›)
proof −
define *f* **where** ‹*f i j x y* ≡ (*if y* < *x then* (*i*::′*a*, *j*::′*a*, *x*::′*b*, *y*::′*b*) *else* (*j*, *i*, *y*, *x*))› **for** *i j x y*
have ‹*?A i j arr* ≤ ⇓*Id* (*?B i j arr*)› **for** *i arr j*
unfolding *hp-link-def f-def*[*symmetric*]
apply *refine-vcg*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *simp* (*metis option.simps*(*2*))
done
moreover have ‹*?B i j arr* ≤ ⇓*Id* (*?A i j arr*)› **for** *i j arr*
unfolding *hp-link-def case-prod-beta f-def*[*symmetric*]
apply *refine-vcg*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *f-def*)
subgoal by (*auto simp*: *f-def*)
subgoal by *auto*
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
by (*cases arr*) *simp*
done
ultimately show *?thesis* **unfolding** *Down-id-eq* **apply** −
apply (*intro ext*)
apply (*rule antisym*)
apply *assumption+*
done
qed

definition *maybe-mop-hp-update-prev′-imp* **where**
‹*maybe-mop-hp-update-prev′-imp child ch arr* =
(*if child* = *None then RETURN arr else mop-hp-update-prev′-imp* (*the child*) *ch arr*)›

definition *maybe-mop-hp-update-nxt′-imp* **where**
‹*maybe-mop-hp-update-nxt′-imp child ch arr* =
(*if child* = *None then RETURN arr else mop-hp-update-nxt′-imp* (*the child*) *ch arr*)›

definition *maybe-mop-hp-update-child′-imp* **where**
‹*maybe-mop-hp-update-child′-imp child ch arr* =
(*if child* = *None then RETURN arr else mop-hp-update-child′-imp* (*the child*) *ch arr*)›

definition *maybe-mop-hp-update-parent′-imp* **where**
‹*maybe-mop-hp-update-parent′-imp child ch arr* =
(*if child* = *None then RETURN arr else mop-hp-update-parent′-imp* (*the child*) *ch arr*)›

lemma *maybe-mop-hp-update-prev′-imp-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈⟨*nat-rel*⟩*option-rel* ⟹ (*j* ≠ *None* ⟹ *the j* ∈# *fst ys*)
⟹
(*p′*,*p*)∈*R* ⟹
maybe-mop-hp-update-prev′-imp i p′ xs ≤ *SPEC* (λ*a*. (*a*, *maybe-hp-update-prev′ j p ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel*)›
unfolding *maybe-mop-hp-update-prev′-imp-def maybe-hp-update-prev′-def*
by (*refine-vcg mop-hp-update-prev′-imp-spec*) *auto*

lemma *maybe-mop-hp-update-nxt′-imp-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈⟨*nat-rel*⟩*option-rel* ⟹ (*j* ≠ *None* ⟹ *the j* ∈# *fst ys*)
⟹
(*p′*,*p*)∈*R* ⟹
maybe-mop-hp-update-nxt′-imp i p′ xs ≤ *SPEC* (λ*a*. (*a*, *maybe-hp-update-nxt′ j p ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel*)›
unfolding *maybe-mop-hp-update-nxt′-imp-def maybe-hp-update-nxt′-def*
by (*refine-vcg mop-hp-update-nxt′-imp-spec*) *auto*

lemma *maybe-mop-hp-update-parent′-imp-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈⟨*nat-rel*⟩*option-rel* ⟹ (*j* ≠ *None* ⟹ *the j* ∈# *fst ys*)
⟹
(*p′*,*p*)∈*R* ⟹
maybe-mop-hp-update-parent′-imp i p′ xs ≤ *SPEC* (λ*a*. (*a*, *maybe-hp-update-parents′ j p ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel*)›
unfolding *maybe-mop-hp-update-parent′-imp-def maybe-hp-update-parents′-def*

by (*refine-vcg mop-hp-update-parent′-imp-spec*) *auto*

lemma *maybe-mop-hp-update-child′-imp-spec*:
‹(*xs*, *ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel* ⟹ (*i*,*j*)∈⟨*nat-rel*⟩*option-rel* ⟹ (*j* ≠ *None* ⟹ *the j* ∈# *fst ys*) ⟹
(*p′*,*p*)∈*R* ⟹
maybe-mop-hp-update-child′-imp i p′ xs ≤ *SPEC* (λ*a*. (*a*, *maybe-hp-update-child′ j p ys*) ∈ ⟨*R*,*S*⟩*pairing-heaps-rel*)›
unfolding *maybe-mop-hp-update-child′-imp-def maybe-hp-update-child′-def*
by (*refine-vcg mop-hp-update-child′-imp-spec*) *auto*

definition *mop-hp-link-imp* :: ‹*nat* ⇒*nat* ⇒(*nat*, ′*b*::*ord*)*pairing-heaps-imp* ⇒ - *nres*› **where**
‹*mop-hp-link-imp* = (λ*i j arr*. *do* {
ASSERT (*i* ≠ *j*);
x ← *mop-hp-read-score-imp i arr*;
y ← *mop-hp-read-score-imp j arr*;
prev ← *mop-hp-read-prev-imp i arr*;
nxt ← *mop-hp-read-nxt-imp j arr*;
let (*parent*,*ch*,w_p, w_{ch}) = (*if y* < *x then* (*i*, *j*, *x*, *y*) *else* (*j*, *i*, *y*, *x*));
child ← *mop-hp-read-child-imp parent arr*;
$child_{ch}$ ← *mop-hp-read-child-imp ch arr*;
arr ← *mop-hp-set-all-imp parent prev nxt* (*Some ch*) *None* ((w_p)) *arr*;
arr ← *mop-hp-set-all-imp ch None child* $child_{ch}$ (*Some parent*) ((w_{ch})) *arr*;
arr ← (*if child* = *None then RETURN arr else mop-hp-update-prev′-imp* (*the child*) (*Some ch*) *arr*);
arr ← (*if nxt* = *None then RETURN arr else mop-hp-update-prev′-imp* (*the nxt*) (*Some parent*) *arr*);
arr ← (*if prev* = *None then RETURN arr else mop-hp-update-nxt′-imp* (*the prev*) (*Some parent*) *arr*);
arr ← (*if child* = *None then RETURN arr else mop-hp-update-parent′-imp* (*the child*) *None arr*);
RETURN (*arr*, *parent*)
})›

lemma *mop-hp-link-imp-spec*:
assumes ‹(*xs*, *ys*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*› ‹(*i*,*j*)∈*nat-rel*› ‹(*w*,*w′*)∈*nat-rel*›
shows ‹*mop-hp-link-imp i w xs* ≤ ⇓(⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel* $\times_r$ *nat-rel*) (*hp-link j w′ ys*)›
proof −
have [*refine*]: ‹(*Some i*, *Some j*) ∈ ⟨*nat-rel*⟩*option-rel*›
using *assms* **by** *auto*
define *f* **where** ‹*f i j x y* ≡ *RETURN* (*if y* < *x then* (*i*::*nat*, *j*::*nat*, *x*::*nat*, *y*::*nat*) *else* (*j*, *i*, *y*, *x*))› **for** *i j x y*
have *Hf*: ‹*do* {*let* (*parent*, *ch*, w_p, w_{ch}) = (*if y* < *x then* (*i*, *j*, *x*, *y*) *else* (*j*, *i*, *y*, *x*)); *P parent ch* w_p w_{ch}} =
do {(*parent*, *ch*, w_p, w_{ch}) ← *f i j x y*; *P parent ch* w_p w_{ch}}› **for** *i j x y w xs P*
unfolding *f-def let-to-bind-conv* **..**

have *K*: ‹*hp-read-child′* (*the* (*source-node ys*)) *ys* ≠ *None* ⟶
the (*hp-read-child′* (*the* (*source-node ys*)) *ys*) ∈ 𝒱 ⟹ *the* (*source-node ys*) ∈# *fst ys* ⟹
op-hp-read-child-imp (*the* (*source-node-impl xs*)) *xs* ≠ *None* ⟹
the (*op-hp-read-child-imp* (*the* (*source-node ys*)) *xs*) ∈ 𝒱› **for** 𝒱
using *op-hp-read-child-imp-spec*[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*› ‹*the* (*source-node ys*)› ‹*the* (*source-node-impl xs*)›]
source-node-spec[*of xs ys* ‹⟨*nat-rel*⟩*option-rel*› ‹⟨*nat-rel*⟩*option-rel*›] *assms*
by *auto*
have [*refine*]: ‹(*x*,*x′*)∈*nat-rel* ⟹ (*y*,*y′*)∈*nat-rel*⟹
f i w x y ≤ ⇓ (*nat-rel* $\times_r$ *nat-rel* $\times_r$ *nat-rel* $\times_r$ *nat-rel*) (*f j w′ x′ y′*)› **for** *x′ y′ x y*
using *assms* **by** *auto*
show *?thesis*

using *assms*
op-hp-read-nxt-imp-spec[*OF assms*(*1*) *assms*(*2*)]
op-hp-read-prev-imp-spec[*OF assms*(*1*) *assms*(*2*)]
op-hp-read-child-imp-spec[*OF assms*(*1*) *assms*(*2*)]
op-hp-read-nxt-imp-spec[*OF assms*(*1*) *assms*(*3*)]
unfolding *mop-hp-link-imp-def hp-link-alt-def Hf*
maybe-mop-hp-update-parent′-imp-def[*symmetric*]
maybe-mop-hp-update-prev′-imp-def[*symmetric*]
maybe-mop-hp-update-nxt′-imp-def[*symmetric*]
maybe-hp-update-prev′-def[*symmetric*]
maybe-hp-update-parents′-def[*symmetric*]
maybe-hp-update-nxt′-def[*symmetric*]
apply −
apply (*refine-vcg mop-hp-set-all-imp-spec*[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-read-score-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*› **and** *ys*=*ys* **and** *i*=*i* **and** *j*=*j*]
mop-hp-read-score-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*› **and** *ys*=*ys* **and** *i*=*w* **and** *j*=*w′*]
Some-x-y-option-theD[**where** *S*=*nat-rel*]
mop-hp-update-parent′-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-update-prev′-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*› **and** *j*=‹*the* (*hp-read-child′* (*the* (*source-node ys*)) *ys*)›]
maybe-mop-hp-update-parent′-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
maybe-mop-hp-update-prev′-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
maybe-mop-hp-update-nxt′-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-read-child-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-read-prev-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-read-nxt-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-read-parent-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›])
subgoal by (*auto dest*: *source-node-spec*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*solves auto*)
subgoal by *auto*
subgoal by *auto*
apply (*solves auto*)
subgoal by *simp*
apply (*solves auto*)
apply (*solves auto*)
done
qed

lemma *vsids-pass*$_1$*-alt-def*:

‹*vsids-pass*$_1$ = (λ(*arr*::′*a multiset* × (′*a*,′*c*::*order*) *hp-fun* × ′*a option*) (*j*::′*a*). *do* {
(*arr*, *j*, -, *n*) ← *WHILE*$_T$(λ(*arr*, *j*,-, -). *j* ≠ *None*)
(λ(*arr*, *j*, *e*::*nat*, *n*). *do* {
if j = *None then RETURN* (*arr*, *None*, *e*, *n*)
else do {
let j = *the j*;
ASSERT (*j* ∈# *fst arr*);
let nxt = *hp-read-nxt*′ *j arr*;
if nxt = *None then RETURN* (*arr*, *nxt*, *e*+*1*, *j*)
else do {
ASSERT (*nxt* ≠ *None*);
ASSERT (*the nxt* ∈# *fst arr*);
let nnxt = *hp-read-nxt*′ (*the nxt*) *arr*;
(*arr*, *n*) ← *hp-link j* (*the nxt*) *arr*;
RETURN (*arr*, *nnxt*, *e*+*2*, *n*)
}}
})
(*arr*, *Some j*, *0*::*nat*, *j*);
RETURN (*arr*, *n*)
})› (**is** ‹*?A* = *?B*›)
proof −
have *K*[*refine*]: ‹*x2* = (*x1a*, *x2a*) ⟹ *i* = (*x1*, *x2*) ⟹
(((*x1*, *x1a*, *x2a*), *Some arr*, *0*, *arr*), *i*::′*a multiset* × (′*a*,′*c*::*order*) *hp-fun* × ′*a option*, *Some arr*, *0*::*nat*, *arr*)
∈ *Id* ×$_r$ ⟨*Id*⟩*option-rel* ×$_r$ *nat-rel* ×$_r$ *Id*›
‹⋀*x1 x2 x1a x2a*.
x2 = (*x1a*, *x2a*) ⟹
i = (*x1*, *x2*) ⟹
((*i*, *Some arr*, *0*, *arr*), (*x1*, *x1a*, *x2a*), *Some arr*, *0*, *arr*)
∈ *Id* ×$_r$ ⟨*Id*⟩*option-rel* ×$_r$ *nat-rel* ×$_r$ *Id*›

for *x2 x1a x2a arr x1 i*
by *auto*
have [*refine*]: ‹(*a*,*a*′)∈*Id* ⟹ (*b*,*b*′)∈*Id* ⟹ (*c*,*c*′)∈*Id* ⟹ *hp-link a b c* ≤⇓*Id* (*hp-link a*′ *b*′ *c*′)› **for** *a b c a*′ *b*′ *c*′
by *auto*
have ‹*?A i arr* ≤ ⇓*Id* (*?B i arr*)› **for** *i arr*
unfolding *vsids-pass*$_1$*-def*
by *refine-vcg* (*solves auto*)+
moreover have ‹*?B i arr* ≤ ⇓*Id* (*?A i arr*)› **for** *i arr*
unfolding *vsids-pass*$_1$*-def*
by *refine-vcg* (*solves* ‹*auto intro*!: *ext bind-cong*[*OF refl*] *simp*: *Let-def*›)+
ultimately show *?thesis* **unfolding** *Down-id-eq* **apply** −
apply (*intro ext*)
apply (*rule antisym*)
apply *assumption*+
done
qed

definition *mop-vsids-pass*$_1$*-imp* :: ‹(*nat*, ′*b*::*ord*)*pairing-heaps-imp* ⇒ *nat* ⇒ - *nres*› **where**
‹*mop-vsids-pass*$_1$*-imp* = (λ*arr j*. *do* {
(*arr*, *j*, *n*) ← *WHILE*$_T$(λ(*arr*, *j*, -). *j* ≠ *None*)
(λ(*arr*, *j*, *n*). *do* {
if j = *None then RETURN* (*arr*, *None*, *n*)
else do {
let j = *the j*;

```
    nxt ← mop-hp-read-nxt-imp j arr;
    if nxt = None then RETURN (arr, nxt, j)
    else do {
      ASSERT (nxt ≠ None);
      nnxt ← mop-hp-read-nxt-imp (the nxt) arr;
      (arr, n) ← mop-hp-link-imp j (the nxt) arr;
      RETURN (arr, nnxt, n)
   }}
  })
  (arr, Some j, j);
  RETURN (arr, n)
 })›
```

```
lemma mop-vsids-pass1-imp-spec:
  assumes ‹(xs, ys) ∈ ⟨⟨nat-rel⟩option-rel,⟨nat-rel⟩option-rel⟩pairing-heaps-rel› ‹(i,j)∈nat-rel›
 shows ‹mop-vsids-pass1-imp xs i ≤ ⇓(⟨⟨nat-rel⟩option-rel,⟨nat-rel⟩option-rel⟩pairing-heaps-rel ×r nat-rel)
(vsids-pass1 ys j)›
proof -
 let ?R = ‹{((arr, j, n), (arr', j', -, n')). (arr, arr') ∈ ⟨⟨nat-rel⟩option-rel,⟨nat-rel⟩option-rel⟩pairing-heaps-rel
∧
    (j, j') ∈ ⟨nat-rel⟩option-rel ∧ (n,n')∈Id}›
  have K[refine0]: ‹((xs, Some i, i), ys, Some j, 0, j) ∈ ?R›
    using assms by auto
  show ?thesis
    unfolding mop-vsids-pass1-imp-def vsids-pass1-alt-def
    apply (refine-vcg mop-hp-insert-impl-spec WHILET-refine[where R= ?R]
      mop-hp-read-nxt-imp-spec[where R=‹⟨nat-rel⟩option-rel› and S=‹⟨nat-rel⟩option-rel›]
      mop-hp-link-imp-spec)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    done
qed
```

```
lemma vsids-pass2-alt-def:
  ‹vsids-pass2 = (λarr (j::'a). do {
  ASSERT (j ∈# fst arr);
  let nxt = hp-read-prev' j arr;
  (arr, j, leader, -) ← WHILET(λ(arr, j, leader, e). j ≠ None)
  (λ(arr, j, leader, e::nat). do {
    if j = None then RETURN (arr, None, leader, e)
    else do {
      let j = the j;
      ASSERT (j ∈# fst arr);
```

let nnxt = hp-read-prev′ j arr;
(*arr*, *n*) ← *hp-link j leader arr*;
RETURN (*arr*, *nnxt*, *n*, *e+1*)
}
})
(*arr*, *nxt*, *j*, *1*::*nat*);
RETURN (*update-source-node* (*Some leader*) *arr*)
})› (**is** ‹*?A* = *?B*›)
proof −
have *K*[*refine*]: ‹(((*fst i*, *fst* (*snd i*), *snd* (*snd i*)), *hp-read-prev arr* (*fst* (*snd i*)), *arr*, *1*::*nat*), *i*, *hp-read-prev′ arr i*, *arr*, *1*::*nat*)
∈ *Id* $\times_r$ ⟨*Id*⟩*option-rel* $\times_r$ *Id* $\times_r$ *Id*›
‹((*i*, *hp-read-prev′ arr i*, *arr*, *1*), (*fst i*, *fst* (*snd i*), *snd* (*snd i*)), *hp-read-prev arr* (*fst* (*snd i*)), *arr*, *1*) ∈ *Id* $\times_r$ ⟨*Id*⟩*option-rel* $\times_r$ *Id* $\times_r$ *Id*›
for *i arr*
by *auto*
have [*refine*]: ‹(*a*,*a′*)∈*Id* ⟹ (*b*,*b′*)∈*Id* ⟹ (*c*,*c′*)∈*Id* ⟹ *hp-link a b c* ≤⇓*Id* (*hp-link a′ b′ c′*)› **for** *a b c a′ b′ c′*
by *auto*
have ‹*?A i arr* ≤ ⇓*Id* (*?B i arr*)› **for** *i arr*
unfolding *vsids-pass*$_2$*-def case-prod-beta*[*of - i*] *case-prod-beta*[*of -* ‹*snd i*›]
by *refine-vcg* (*solves auto*)+
moreover have ‹*?B i arr* ≤ ⇓*Id* (*?A i arr*)› **for** *i arr*
unfolding *vsids-pass*$_2$*-def case-prod-beta*[*of - i*] *case-prod-beta*[*of -* ‹*snd i*›]
by *refine-vcg* (*solves* ‹*auto intro*!: *ext bind-cong*[*OF refl*] *simp*: *Let-def*›)+
ultimately show *?thesis* **unfolding** *Down-id-eq* **apply** −
apply (*intro ext*)
apply (*rule antisym*)
apply *assumption*+
done
qed

definition *mop-vsids-pass*$_2$*-imp* **where**
‹*mop-vsids-pass*$_2$*-imp* = (λ*arr* (*j*::*nat*). *do* {
nxt ← *mop-hp-read-prev-imp j arr*;
(*arr*, *j*, *leader*) ← *WHILE*$_T$(λ(*arr*, *j*, *leader*). *j* ≠ *None*)
(λ(*arr*, *j*, *leader*). *do* {
if j = *None then RETURN* (*arr*, *None*, *leader*)
else do {
let j = *the j*;
nnxt ← *mop-hp-read-prev-imp j arr*;
(*arr*, *n*) ← *mop-hp-link-imp j leader arr*;
RETURN (*arr*, *nnxt*, *n*)
}
})
(*arr*, *nxt*, *j*);
RETURN (*update-source-node-impl* (*Some leader*) *arr*)
})›

lemma *mop-vsids-pass*$_2$*-imp-spec*:
assumes ‹(*xs*, *ys*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*› ‹(*i*,*j*)∈*nat-rel*›
shows ‹*mop-vsids-pass*$_2$*-imp xs i* ≤ ⇓(⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*) (*vsids-pass*$_2$ *ys j*)›
proof −
let *?R* = ‹{((*arr*, *j*, *n*), (*arr′*, *j′*, *n′*, -)). (*arr*, *arr′*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel* ∧

$(j, j') \in \langle nat\text{-}rel\rangle option\text{-}rel \wedge (n,n')\in Id\}$›
  have *K*[*refine0*]: ‹((*xs*, *Some i*, *i*), *ys*, *Some j*, *j*, *0*) ∈ *?R*›
    using *assms* **by** *auto*
  show *?thesis*
    using *assms*
    unfolding *mop-vsids-pass*$_2$*-imp-def vsids-pass*$_2$*-alt-def*
    apply (*refine-vcg mop-hp-insert-impl-spec WHILET-refine*[**where** *R*= *?R*]
      mop-hp-read-nxt-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
      mop-hp-read-prev-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
      mop-hp-link-imp-spec mop-vsids-pass$_1$*-imp-spec*
      update-source-node-impl-spec)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    done
qed

definition *mop-merge-pairs-imp* **where**
  ‹*mop-merge-pairs-imp arr j* = *do* {
    (*arr*, *j*) ← *mop-vsids-pass*$_1$*-imp arr j*;
    mop-vsids-pass$_2$*-imp arr j*
  }›

lemma *mop-merge-pairs-imp-spec*:
  assumes ‹(*xs*, *ys*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*› ‹(*i*,*j*)∈*nat-rel*›
  shows ‹*mop-merge-pairs-imp xs i* ≤ ⇓(⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*) (*merge-pairs ys j*)›
  using *assms* **unfolding** *mop-merge-pairs-imp-def merge-pairs-def*
  by (*refine-vcg mop-vsids-pass*$_1$*-imp-spec mop-vsids-pass*$_2$*-imp-spec*) *auto*

lemma *vsids-pop-min-alt-def*:
  ‹*vsids-pop-min* = (λ*arr*. *do* {
  let h = *source-node arr*;
  if h = *None then RETURN* (*None*, *arr*)
  else do {
      ASSERT (*the h* ∈# *fst arr*);
      let j = *hp-read-child'* (*the h*) *arr*;
      if j = *None then RETURN* (*h*, (*update-source-node None arr*))
      else do {
        ASSERT (*the j* ∈# *fst arr*);
        let arr = *hp-update-prev'* (*the h*) *None arr*;
        let arr = *hp-update-child'* (*the h*) *None arr*;
        let arr = *hp-update-parents'* (*the j*) *None arr*;
        arr ← *merge-pairs* (*update-source-node None arr*) (*the j*);
        RETURN (*h*, *arr*)
      }

```
    }
  })› (is ‹?A = ?B›)
proof -
  have [simp]: ‹source-node arr = None ⟹ (fst arr, fst (snd arr), None) = arr› for arr
    by (cases arr) auto
  have K[refine]: ‹((source-node arr, fst arr, fst (snd arr), None), source-node arr,
  update-source-node None arr)
    ∈ Id›
    ‹((source-node arr, update-source-node None arr), source-node arr, fst arr, fst (snd arr), None)
    ∈ Id›
    for i arr
    by (solves ‹cases arr; auto›)+
  have [refine]: ‹merge-pairs
  (fst arr,
   hp-update-parents (the (hp-read-child (the (source-node arr)) (fst (snd arr))))
    None
    (hp-update-child (the (source-node arr)) None
      (hp-update-prev (the (source-node arr)) None (fst (snd arr)))),
   None)
  (the (hp-read-child (the (source-node arr)) (fst (snd arr))))
    ≤ ⇓ Id
    (merge-pairs
      (update-source-node None
     (hp-update-parents' (the (hp-read-child' (the (source-node arr)) arr)) None
       (hp-update-child' (the (source-node arr)) None
      (hp-update-prev' (the (source-node arr)) None arr))))
    (the (hp-read-child' (the (source-node arr)) arr)))›
    ‹merge-pairs
   (update-source-node None
  (hp-update-parents' (the (hp-read-child' (the (source-node arr)) arr)) None
    (hp-update-child' (the (source-node arr)) None
      (hp-update-prev' (the (source-node arr)) None arr))))
   (the (hp-read-child' (the (source-node arr)) arr))
  ≤ ⇓ Id
  (merge-pairs
    (fst arr,
     hp-update-parents (the (hp-read-child (the (source-node arr)) (fst (snd arr)))) None
      (hp-update-child (the (source-node arr)) None
     (hp-update-prev (the (source-node arr)) None (fst (snd arr)))),
     None)
    (the (hp-read-child (the (source-node arr)) (fst (snd arr)))))› for arr
    by (solves ‹cases arr; auto›)+
  have K: ‹snd (snd arr) = source-node arr› for arr
    by (cases arr) auto

  have ‹?A arr ≤ ⇓Id (?B arr)› for i arr
    unfolding vsids-pop-min-def case-prod-beta[of - arr] case-prod-beta[of - ‹snd arr›]
      K
    by refine-vcg (solves auto)+
  moreover have ‹?B arr ≤ ⇓Id (?A arr)› for i arr
    unfolding vsids-pop-min-def case-prod-beta[of - arr] case-prod-beta[of - ‹snd arr›] K
    by refine-vcg (solves ‹auto intro!: ext bind-cong[OF refl] simp: Let-def›)+
  ultimately show ?thesis unfolding Down-id-eq apply -
    apply (intro ext)
    apply (rule antisym)
    apply assumption+
```

done
qed

definition *mop-vsids-pop-min-impl* **where**
‹*mop-vsids-pop-min-impl* = (λ*arr*. *do* {
let h = *source-node-impl arr*;
if h = *None then RETURN* (*None*, *arr*)
else do {
j ← *mop-hp-read-child-imp* (*the h*) *arr*;
if j = *None then RETURN* (*h*, *update-source-node-impl None arr*)
else do {
arr ← *mop-hp-update-prev′-imp* (*the h*) *None arr*;
arr ← *mop-hp-update-child′-imp* (*the h*) *None arr*;
arr ← *mop-hp-update-parent′-imp* (*the j*) *None arr*;
arr ← *mop-merge-pairs-imp* (*update-source-node-impl None arr*) (*the j*);
RETURN (*h*, *arr*)
}
}
})›

lemma *mop-vsids-pop-min-impl*:
assumes ‹(*xs*, *ys*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*›
shows ‹*mop-vsids-pop-min-impl xs* ≤ ⇓(⟨*nat-rel*⟩*option-rel* $\times_r$ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*) (*vsids-pop-min ys*)›
proof −
let *?R* = ‹{((*arr*, *j*, *n*), (*arr′*, *j′*, *n′*, -)). (*arr*, *arr′*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel* ∧
(*j*, *j′*) ∈ ⟨*nat-rel*⟩*option-rel* ∧ (*n*,*n′*)∈*Id*}›
have *K*[*refine0*]: ‹(*the* (*source-node-impl xs*), *the* (*source-node ys*)) ∈ *nat-rel*›
if ‹*source-node ys* ≠ *None*›
using *source-node-spec*[*OF assms*] **by** *auto*
show *?thesis*
using *assms source-node-spec*[*OF assms*]
unfolding *mop-vsids-pop-min-impl-def vsids-pop-min-alt-def*
apply (*refine-vcg mop-hp-insert-impl-spec WHILET-refine*[**where** *R*= *?R*]
mop-hp-read-nxt-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-read-prev-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-link-imp-spec mop-vsids-pass$_1$*-imp-spec*
mop-merge-pairs-imp-spec
mop-hp-read-child-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-update-prev′-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-update-child′-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-update-parent′-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *update-source-node-impl-spec*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *update-source-node-impl-spec*)

```
    subgoal by auto
    subgoal by auto
    done
qed


definition mop-vsids-pop-min2-impl where
    ‹mop-vsids-pop-min2-impl = (λarr. do {
  let h = source-node-impl arr;
  ASSERT (h ≠ None);
  j ← mop-hp-read-child-imp (the h) arr;
  if j = None then RETURN (the h, update-source-node-impl None arr)
  else do {
    arr ← mop-hp-update-prev'-imp (the h) None arr;
    arr ← mop-hp-update-child'-imp (the h) None arr;
    arr ← mop-hp-update-parent'-imp (the j) None arr;
    arr ← mop-merge-pairs-imp (update-source-node-impl None arr) (the j);
    RETURN (the h, arr)
  }
  })›


lemma vsids-pop-min2-alt-def:
  ‹vsids-pop-min2 = (λarr. do {
  let h = source-node arr;
  ASSERT (h ≠ None);
  ASSERT (the h ∈# fst arr);
  let j = hp-read-child' (the h) arr;
  if j = None then RETURN (the h, (update-source-node None arr))
  else do {
    ASSERT (the j ∈# fst arr);
    let arr = hp-update-prev' (the h) None arr;
    let arr = hp-update-child' (the h) None arr;
    let arr = hp-update-parents' (the j) None arr;
    arr ← merge-pairs (update-source-node None arr) (the j);
    RETURN (the h, arr)
    }
  })› (is ‹?A = ?B›)
proof -
  have [simp]: ‹source-node arr = None ⟹ (fst arr, fst (snd arr), None) = arr› for arr
    by (cases arr) auto
  have K[refine]: ‹((the (source-node arr), fst arr, fst (snd arr), None), the (source-node arr),
  update-source-node None arr)
    ∈ Id›
    ‹((the (source-node arr), fst arr, fst (snd arr), None), the (source-node arr),
          update-source-node None arr)
    ∈ Id›
    ‹((the (source-node arr), update-source-node None arr), the (source-node arr), fst arr,
               fst (snd arr), None)
              ∈ Id›
    for i arr
    by (solves ‹cases arr; auto›)+
  have [refine]: ‹merge-pairs
  (fst arr,
   hp-update-parents (the (hp-read-child (the (source-node arr)) (fst (snd arr))))
```

None
(*hp-update-child* (*the* (*source-node arr*)) *None*
(*hp-update-prev* (*the* (*source-node arr*)) *None* (*fst* (*snd arr*)))),
None)
(*the* (*hp-read-child* (*the* (*source-node arr*)) (*fst* (*snd arr*))))
≤ ⇓ *Id*
(*merge-pairs*
(*update-source-node None*
(*hp-update-parents'* (*the* (*hp-read-child'* (*the* (*source-node arr*)) *arr*)) *None*
(*hp-update-child'* (*the* (*source-node arr*)) *None*
(*hp-update-prev'* (*the* (*source-node arr*)) *None arr*))))
(*the* (*hp-read-child'* (*the* (*source-node arr*)) *arr*)))›
‹*merge-pairs*
(*update-source-node None*
(*hp-update-parents'* (*the* (*hp-read-child'* (*the* (*source-node arr*)) *arr*)) *None*
(*hp-update-child'* (*the* (*source-node arr*)) *None*
(*hp-update-prev'* (*the* (*source-node arr*)) *None arr*))))
(*the* (*hp-read-child'* (*the* (*source-node arr*)) *arr*))
≤ ⇓ *Id*
(*merge-pairs*
(*fst arr*,
hp-update-parents (*the* (*hp-read-child* (*the* (*source-node arr*)) (*fst* (*snd arr*)))) *None*
(*hp-update-child* (*the* (*source-node arr*)) *None*
(*hp-update-prev* (*the* (*source-node arr*)) *None* (*fst* (*snd arr*)))),
None)
(*the* (*hp-read-child* (*the* (*source-node arr*)) (*fst* (*snd arr*)))))› **for** *arr*
by (*solves* ‹*cases arr*; *auto*›)+
have *K*: ‹*snd* (*snd arr*) = *source-node arr*› **for** *arr*
by (*cases arr*) *auto*

have ‹*?A arr* ≤ ⇓*Id* (*?B arr*)› **for** *i arr*
unfolding *vsids-pop-min2-def case-prod-beta*[*of - arr*] *case-prod-beta*[*of -* ‹*snd arr*›] *Let-def*
K
by *refine-vcg* (*solves auto*)+
moreover have ‹*?B arr* ≤ ⇓*Id* (*?A arr*)› **for** *i arr*
unfolding *vsids-pop-min2-def case-prod-beta*[*of - arr*] *case-prod-beta*[*of -* ‹*snd arr*›] *K*
by *refine-vcg* (*solves* ‹*auto intro*!: *ext bind-cong*[*OF refl*] *simp*: *Let-def*›)+
ultimately show *?thesis* **unfolding** *Down-id-eq* **apply** −
apply (*intro ext*)
apply (*rule antisym*)
apply *assumption*+
done
qed

lemma *mop-vsids-pop-min2-impl*:
assumes ‹(*xs*, *ys*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*›
shows ‹*mop-vsids-pop-min2-impl xs* ≤ ⇓(*nat-rel* $\times_r$ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*)
(*vsids-pop-min2 ys*)›
proof −
let *?R* = ‹{((*arr*, *j*, *n*), (*arr'*, *j'*, *n'*, -)). (*arr*, *arr'*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*
∧
(*j*, *j'*) ∈ ⟨*nat-rel*⟩*option-rel* ∧ (*n*,*n'*)∈*Id*}›
have *K*[*refine0*]: ‹(*the* (*source-node-impl xs*), *the* (*source-node ys*)) ∈ *nat-rel*›
if ‹*source-node ys* ≠ *None*›
using *source-node-spec*[*OF assms*] **by** *auto*
show *?thesis*

using *assms source-node-spec*[*OF assms*]
unfolding *mop-vsids-pop-min2-impl-def vsids-pop-min2-alt-def*
apply (*refine-vcg mop-hp-insert-impl-spec WHILET-refine*[**where** *R*= *?R*]
mop-hp-read-nxt-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-read-prev-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-link-imp-spec mop-vsids-pass$_1$*-imp-spec*
mop-merge-pairs-imp-spec
mop-hp-read-child-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-update-prev'-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-update-child'-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-update-parent'-imp-spec[**where** *R*=‹⟨*nat-rel*⟩*option-rel*› **and** *S*=‹⟨*nat-rel*⟩*option-rel*›])
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *update-source-node-impl-spec*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *update-source-node-impl-spec*)
subgoal by *auto*
subgoal by *auto*
done
qed

definition *mop-unroot-hp-tree* **where**
‹*mop-unroot-hp-tree arr h* = *do* {
let a = *source-node-impl arr*;
nnext ← *mop-hp-read-nxt-imp h arr*;
parent ← *mop-hp-read-parent-imp h arr*;
prev ← *mop-hp-read-prev-imp h arr*;
if prev = *None* ∧ *parent* = *None* ∧ *Some h* ≠ *a then RETURN* (*update-source-node-impl None arr*)
else if Some h = *a then RETURN* (*update-source-node-impl None arr*)
else do {
ASSERT (*a* ≠ *None*);
let a' = *the a*;
arr ← *maybe-mop-hp-update-child'-imp parent nnext arr*;
arr ← *maybe-mop-hp-update-nxt'-imp prev nnext arr*;
arr ← *maybe-mop-hp-update-prev'-imp nnext prev arr*;
arr ← *maybe-mop-hp-update-parent'-imp nnext parent arr*;

arr ← *mop-hp-update-nxt'-imp h None arr*;
arr ← *mop-hp-update-prev'-imp h None arr*;
arr ← *mop-hp-update-parent'-imp h None arr*;

arr ← *mop-hp-update-nxt'-imp h* (*Some a'*) *arr*;
arr ← *mop-hp-update-prev'-imp a'* (*Some h*) *arr*;
RETURN (*update-source-node-impl None arr*)
}
}›

lemma *mop-unroot-hp-tree-spec*:
assumes ‹(*xs*, *ys*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*› **and** ‹(*h*,*i*)∈*nat-rel*›
shows ‹*mop-unroot-hp-tree xs h* ≤ ⇓(⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*) (*unroot-hp-tree ys i*)›

proof −
show *?thesis*
using *assms* **using** *source-node-spec*[*OF assms*(*1*)]
unfolding *mop-unroot-hp-tree-def unroot-hp-tree-def*
apply (*refine-rcg mop-hp-read-nxt-imp-spec assms*
mop-hp-read-parent-imp-spec mop-hp-read-prev-imp-spec
update-source-node-impl-spec maybe-mop-hp-update-child′-imp-spec
maybe-mop-hp-update-nxt′-imp-spec maybe-mop-hp-update-parent′-imp-spec
maybe-mop-hp-update-prev′-imp-spec
mop-hp-update-nxt′-imp-spec[**where** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-update-prev′-imp-spec[**where** *S*=‹⟨*nat-rel*⟩*option-rel*›]
mop-hp-update-parent′-imp-spec[**where** *S*=‹⟨*nat-rel*⟩*option-rel*›])
subgoal using *source-node-spec*[*OF assms*(*1*)] **by** *auto*
subgoal by *auto*
subgoal using *source-node-spec*[*OF assms*(*1*)] **by** *auto*
subgoal by *auto*
subgoal using *source-node-spec*[*OF assms*(*1*)] **by** *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *mop-rescale-and-reroot* **where**
‹*mop-rescale-and-reroot h w′ arr = do* {
nnext ← mop-hp-read-nxt-imp h arr;
parent ← mop-hp-read-parent-imp h arr;
prev ← mop-hp-read-prev-imp h arr;
if source-node-impl arr = None then mop-hp-update-score-imp h w′ arr
else if prev = None ∧ parent = None ∧ Some h ≠ source-node-impl arr then mop-hp-update-score-imp h w′ arr

```
    else if Some h = source-node-impl arr then mop-hp-update-score-imp h w′ arr
    else do {
       arr ← mop-unroot-hp-tree arr h;
       arr ← mop-hp-update-score-imp h w′ arr;
       mop-merge-pairs-imp arr h
    }
})›


lemma mop-rescale-and-reroot-spec:
  assumes ‹(xs, ys) ∈ ⟨⟨nat-rel⟩option-rel,⟨nat-rel⟩option-rel⟩pairing-heaps-rel› and ‹(h,i)∈nat-rel› ‹(w, w′) ∈ nat-rel›
  shows ‹mop-rescale-and-reroot h w xs ≤ ⇓(⟨⟨nat-rel⟩option-rel,⟨nat-rel⟩option-rel⟩pairing-heaps-rel) (rescale-and-reroot i w′ ys)›
proof −
  have [refine]: ‹(Some w, Some w′) ∈ ⟨nat-rel⟩option-rel›
    using assms by auto
  show ?thesis
    using source-node-spec[OF assms(1)] assms(2,3)
    unfolding rescale-and-reroot-def mop-rescale-and-reroot-def
    apply (refine-rcg mop-hp-read-nxt-imp-spec assms mop-hp-read-parent-imp-spec
      mop-hp-read-prev-imp-spec mop-hp-update-score-imp-spec mop-unroot-hp-tree-spec
      mop-merge-pairs-imp-spec)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    apply assumption
    subgoal by auto
    done
qed

definition mop-hp-is-in :: ‹-› where
  ‹mop-hp-is-in h = (λarr. do {
   parent ← mop-hp-read-parent-imp h arr;
   prev ← mop-hp-read-prev-imp h arr;
   let s = source-node-impl arr;
   RETURN (s ≠ None ∧ (prev ≠ None ∨ parent ≠ None ∨ the s = h))
  })›


lemma mop-hp-is-in-spec:
  assumes ‹(xs, ys) ∈ ⟨⟨nat-rel⟩option-rel,⟨nat-rel⟩option-rel⟩pairing-heaps-rel› and ‹(h,i)∈nat-rel›
  shows ‹mop-hp-is-in h xs ≤ ⇓bool-rel (hp-is-in i ys)›
proof −
  have hp-is-in-alt-def: ‹hp-is-in w = (λbw. do {
  ASSERT (w ∈# fst bw);
  let parent = hp-read-parent′ w bw;
  let prev = hp-read-prev′ w bw;
   let s = source-node bw;
  RETURN (s ≠ None ∧ (hp-read-prev′ w bw ≠ None ∨ hp-read-parent′ w bw ≠ None ∨ the s = w))
  })› for w
  by (auto simp: hp-is-in-def)
  show ?thesis
    using source-node-spec[OF assms(1)] assms(2)
    unfolding mop-hp-is-in-def hp-is-in-alt-def
    by (refine-vcg assms mop-hp-read-parent-imp-spec mop-hp-read-prev-imp-spec)
```

auto
qed

lemma *mop-hp-read-score-imp-mop-hp-read-score*:
assumes ‹(*xs*, *ys*) ∈ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*› **and** ‹(*h*,*i*)∈*nat-rel*›
shows ‹*mop-hp-read-score-imp h xs* ≤ ⇓*nat-rel* (*mop-hp-read-score i ys*)›
unfolding *mop-hp-read-score-def case-prod-beta mop-hp-read-score-imp-def*
apply (*refine-vcg mop-hp-read-score-imp-spec*)
using *assms* **apply** (*auto simp*: *pairing-heaps-rel-def map-fun-rel-def dest*!: *multi-member-split*)
done

end
end