Finite Map Extras

Javier Díaz
<javier.diaz.manzi@gmail.com>

September 13, 2023

Abstract

This includes useful syntactic sugar, new operators and functions and their associated lemmas for finite maps which currently are not present in the standard `Finite_Map` theory.

Contents

1 Extra Features for Finite Maps **1**

1 Extra Features for Finite Maps

theory *Finite-Map-Extras*
imports *HOL−Library.Finite-Map*
begin

Extra lemmas and syntactic sugar for *fmap*

notation *fmlookup* (**infixl** ‹$$› *900*)

notation *fmempty* (‹{$$}›)

nonterminal *fmaplets* **and** *fmaplet*

syntax
-fmaplet :: ['*a*, '*a*] ⇒ *fmaplet* (- /$$:=/ -)
-fmaplets :: ['*a*, '*a*] ⇒ *fmaplet* (- /[$$:=]/ -)
:: *fmaplet* ⇒ *fmaplets* (-)
-Fmaplets :: [*fmaplet*, *fmaplets*] ⇒ *fmaplets* (-,/ -)
-FmapUpd :: [('*a*, '*b*) *fmap*, *fmaplets*] ⇒ ('*a*, '*b*) *fmap* (-/'(-') [*900*, *0*] *900*)
-Fmap :: *fmaplets* ⇒ ('*a*, '*b*) *fmap* ((*1*{-}))

translations
-FmapUpd m (*-Fmaplets xy ms*) ⇌ *-FmapUpd* (*-FmapUpd m xy*) *ms*
-FmapUpd m (*-fmaplet x y*) ⇌ *CONST fmupd x y m*
-Fmap ms ⇌ *-FmapUpd* (*CONST fmempty*) *ms*
-Fmap (*-Fmaplets ms1 ms2*) ↽ *-FmapUpd* (*-Fmap ms1*) *ms2*

-Fmaplets ms1 (-Fmaplets ms2 ms3) ↼ *-Fmaplets (-Fmaplets ms1 ms2) ms3*

abbreviation *fmap-lookup-the* (**infixl** ‹$$!› *900*) **where**
m $$! *k* ≡ *the* (*m* $$ *k*)

lemma *fmadd-singletons-comm*:
assumes $k_1 \neq k_2$
shows $\{k_1 \$\$:= v_1\} +\!\!+_f \{k_2 \$\$:= v_2\} = \{k_2 \$\$:= v_2\} +\!\!+_f \{k_1 \$\$:= v_1\}$
proof (*intro fmap-ext*)
fix *k*
consider
(*a*) $k = k_1$ |
(*b*) $k = k_2$ |
(*c*) $k \neq k_1 \wedge k \neq k_2$
by *auto*
with *assms* **show** $(\{k_1 \$\$:= v_1\} +\!\!+_f \{k_2 \$\$:= v_2\})\ \$\$\ k = (\{k_2 \$\$:= v_2\} +\!\!+_f \{k_1 \$\$:= v_1\})\ \$\$\ k$
by *auto*
qed

lemma *fmap-singleton-comm*:
assumes *m* $$ *k* = *None*
shows $m +\!\!+_f \{k \$\$:= v\} = \{k \$\$:= v\} +\!\!+_f m$
using *assms*
proof (*induction m arbitrary*: *k v*)
case *fmempty*
then show *?case*
by *simp*
next
case (*fmupd x y m*)
have $m(x \$\$:= y) +\!\!+_f \{k \$\$:= v\} = m +\!\!+_f \{x \$\$:= y\} +\!\!+_f \{k \$\$:= v\}$
by *simp*
also from *fmupd.hyps* **and** *fmupd.IH* **have** $\ldots = \{x \$\$:= y\} +\!\!+_f m +\!\!+_f \{k \$\$:= v\}$
by *simp*
also from *fmupd.prems* **and** *fmupd.hyps* **and** *fmupd.IH* **have** $\ldots = \{x \$\$:= y\} +\!\!+_f \{k \$\$:= v\} +\!\!+_f m$
by (*metis fmadd-assoc fmupd-lookup*)
also have $\ldots = \{k \$\$:= v\} +\!\!+_f m(x \$\$:= y)$
proof (*cases* $x \neq k$)
case *True*
then have $\{x \$\$:= y\} +\!\!+_f \{k \$\$:= v\} +\!\!+_f m = \{k \$\$:= v\} +\!\!+_f \{x \$\$:= y\} +\!\!+_f m$
using *fmadd-singletons-comm* **by** *metis*
also from *fmupd.prems* **and** *fmupd.hyps* **and** *fmupd.IH* **have** $\ldots = \{k \$\$:= v\} +\!\!+_f m +\!\!+_f \{x \$\$:= y\}$
by (*metis fmadd-assoc*)
finally show *?thesis*
by *simp*
next
case *False*
with *fmupd.prems* **show** *?thesis*

```
    by auto
  qed
  finally show ?case .
qed

lemma fmap-disj-comm:
  assumes fmdom' m1 ∩ fmdom' m2 = {}
  shows m1 ++f m2 = m2 ++f m1
  using assms
proof (induction m2 arbitrary: m1)
  case fmempty
  then show ?case
    by simp
next
  case (fmupd k v m2)
  then show ?case
  proof (cases m1 $$ k = None)
    case True
    from fmupd.hyps have m1 ++f m2(k $$:= v) = m1 ++f m2 ++f {k $$:= v}
      by simp
    also from fmupd.prems and fmupd.hyps and fmupd.IH have ... = m2 ++f m1 ++f {k $$:= v}
      by simp
    also from True have ... = m2 ++f {k $$:= v} ++f m1
      using fmap-singleton-comm by (metis fmadd-assoc)
    finally show ?thesis
      by simp
  next
    case False
    with fmupd.prems show ?thesis
      by auto
  qed
qed

lemma fmran-singleton: fmran {k $$:= v} = {|v|}
proof −
  have v' |∈| fmran {k $$:= v} ⟹ v' = v for v'
  proof −
    assume v' |∈| fmran {k $$:= v}
    fix k'
    have fmdom' {k $$:= v} = {k}
      by simp
    then show v' = v
    proof (cases k' = k)
      case True
      with ‹v' |∈| fmran {k $$:= v}› show ?thesis
        using fmdom'I by fastforce
    next
      case False
      with ‹fmdom' {k $$:= v} = {k}› and ‹v' |∈| fmran {k $$:= v}› show ?thesis
```

```
      using fmdom'I by fastforce
    qed
  qed
  moreover have v |∈| fmran {k $$:= v}
    by (simp add: fmranI)
  ultimately show ?thesis
    by (simp add: fsubsetI fsubset-antisym)
qed

lemma fmmap-keys-hom:
  assumes fmdom' m1 ∩ fmdom' m2 = {}
  shows fmmap-keys f (m1 ++f m2) = fmmap-keys f m1 ++f fmmap-keys f m2
  using assms
  by (simp add: fmap-ext)

lemma map-insort-is-insort-key:
  assumes m $$ k = None
  shows map (λk'. (k', m(k $$:= v) $$! k')) (insort k xs) =
    insort-key fst (k, v) (map (λk'. (k', m(k $$:= v) $$! k')) xs)
  using assms by (induction xs) auto

lemma sorted-list-of-fmap-is-insort-key-fst:
  assumes m $$ k = None
  shows sorted-list-of-fmap (m(k $$:= v)) = insort-key fst (k, v) (sorted-list-of-fmap m)
proof −
  have sorted-list-of-fmap (m(k $$:= v)) =
    map (λk'. (k', m(k $$:= v) $$! k')) (sorted-list-of-fset (fmdom (m(k $$:= v))))
    unfolding sorted-list-of-fmap-def ..
  also have ... =  map (λk'. (k', m(k $$:= v) $$! k')) (sorted-list-of-fset (finsert k (fmdom m)))
    by simp
  also from ‹m $$ k = None› have ... =
    map (λk'. (k', m(k $$:= v) $$! k')) (insort k (sorted-list-of-fset (fmdom m − {|k|})))
    by (simp add: sorted-list-of-fset.rep-eq)
  also from ‹m $$ k = None› have ... =
    map (λk'. (k', m(k $$:= v) $$! k')) (insort k (sorted-list-of-fset (fmdom m)))
    by (simp add: fmdom-notI)
  also from ‹m $$ k = None› have ... =
    insort-key fst (k, v) (map (λk'. (k', m(k $$:= v) $$! k')) (sorted-list-of-fset (fmdom m)))
    using map-insort-is-insort-key by fastforce
  also have ... = insort-key fst (k, v) (map (λk'. (k', m $$! k')) (sorted-list-of-fset (fmdom m)))
  proof −
    from ‹m $$ k = None› have ⋀k'. k' ∈ fmdom' m ⟹ m(k $$:= v) $$! k' = m $$! k'
      using fmdom'-notI by force
    moreover from ‹m $$ k = None› have k ∉ set (sorted-list-of-fset (fmdom m))
      using fmdom'-alt-def and fmdom'-notI and in-set-member by force
    ultimately show ?thesis
      by (metis (mono-tags, lifting) fmdom'-alt-def map-eq-conv sorted-list-of-fset-simps(1))
  qed
  finally show ?thesis
```

```
    unfolding sorted-list-of-fmap-def by simp
qed

lemma distinct-fst-inj:
  assumes distinct (map fst ps)
  and inj f
  shows distinct (map fst (map (λ(k, v). (f k, v)) ps))
proof −
  have map fst (map (λ(k, v). (f k, v)) ps) = map f (map fst ps)
    by (induction ps) auto
  moreover from assms have distinct (map f (map fst ps))
    by (simp add: distinct-map inj-on-def)
  ultimately show ?thesis
    by presburger
qed

lemma distinct-sorted-list-of-fmap:
  shows distinct (map fst (sorted-list-of-fmap m))
  unfolding sorted-list-of-fmap-def and sorted-list-of-fset-def
  by (simp add: distinct-map inj-on-def)

lemma map-inj-pair-non-membership:
  assumes k ∉ set (map fst ps)
  and inj f
  shows f k ∉ set (map fst (map (λ(k, v). (f k, v)) ps))
  using assms by (induction ps) (simp add: member-rec(2), fastforce simp add: injD)

lemma map-insort-key-fst:
  assumes distinct (map fst ps)
  and k ∉ set (map fst ps)
  and inj f
  and mono f
  shows map (λ(k, v). (f k, v)) (insort-key fst (k, v) ps) =
    insort-key fst (f k, v) (map (λ(k, v). (f k, v)) ps)
  using assms
proof (induction ps)
  case Nil
  then show ?case
    by simp
next
  let ?g = (λ(k, v). (f k, v))
  case (Cons p ps)
  then show ?case
  proof (cases k ≤ fst p)
    case True
    let ?f-p = (f (fst p), snd p)
    have insort-key fst (f k, v) (map ?g (p # ps)) = insort-key fst (f k, v) (?f-p # map ?g ps)
      by (simp add: prod.case-eq-if)
    moreover from Cons.prems(4) and True have f k ≤ f (fst p)
```

```
    by (auto dest: monoE)
   then have insort-key fst (f k, v) (?f-p # map ?g ps) = (f k, v) # ?f-p # map ?g ps
    by simp
   ultimately have insort-key fst (f k, v) (map ?g (p # ps)) = (f k, v) # ?f-p # map ?g ps
    by simp
  moreover from True have map ?g (insort-key fst (k, v) (p # ps)) = (f k, v) # ?f-p # map ?g ps
    by (simp add: case-prod-beta')
   ultimately show ?thesis
    by simp
 next
   case False
   let ?f-p = (f (fst p), snd p)
   have insort-key fst (f k, v) (map ?g (p # ps)) = insort-key fst (f k, v) (?f-p # map ?g ps)
    by (simp add: prod.case-eq-if)
   moreover from ‹mono f› and False have f (fst p) ≤ f k
    using not-le by (blast dest: mono-invE)
   ultimately have insort-key fst (f k, v) (map ?g (p # ps)) =
    ?f-p # insort-key fst (f k, v) (map ?g ps)
    using False and ‹inj f› by (fastforce dest: injD)
   also from Cons.IH and Cons.prems(1,2) and assms(3,4) have ... =
    ?f-p # (map ?g (insort-key fst (k, v) ps))
    by (fastforce simp add: member-rec(1))
   also have ... = map ?g (p # insort-key fst (k, v) ps)
    by (simp add: case-prod-beta)
   finally show ?thesis
    using False by simp
 qed
qed

lemma map-sorted-list-of-fmap:
 assumes inj f
 and mono f
 and m $$ k = None
 shows map (λ(k, v). (f k, v)) (sorted-list-of-fmap (m(k $$:= v))) =
   insort-key fst (f k, v) (map (λ(k, v). (f k, v)) (sorted-list-of-fmap m))
proof −
 let ?g = (λ(k, v). (f k, v))
 from ‹m $$ k = None› have map ?g (sorted-list-of-fmap (m(k $$:= v))) =
  map ?g (insort-key fst (k, v) (sorted-list-of-fmap m))
  using sorted-list-of-fmap-is-insort-key-fst by fastforce
 also have ... = insort-key fst (f k, v) (map ?g (sorted-list-of-fmap m))
 proof −
  have distinct (map fst (sorted-list-of-fmap m))
   by (simp add: distinct-sorted-list-of-fmap)
  moreover from ‹m $$ k = None› have k ∉ set (map fst (sorted-list-of-fmap m))
   by (metis image-set map-of-eq-None-iff map-of-sorted-list)
  ultimately show ?thesis
   by (simp add: map-insort-key-fst assms(1,2))
 qed
```

```
  finally show ?thesis .
qed

lemma fmap-of-list-insort-key-fst:
  assumes distinct (map fst ps)
  and k ∉ set (map fst ps)
  shows fmap-of-list (insort-key fst (k, v) ps) = (fmap-of-list ps)(k $$:= v)
  using assms
proof (induction ps)
  case Nil
  then show ?case
    by simp
next
  case (Cons p ps)
  then show ?case
  proof (cases k ≤ fst p)
    case True
    then show ?thesis
      by simp
  next
    case False
    then have fmap-of-list (insort-key fst (k, v) (p # ps)) =
      fmap-of-list (p # insort-key fst (k, v) ps)
      by simp
    also have ... = (fmap-of-list (insort-key fst (k, v) ps))(fst p $$:= snd p)
      by (metis fmap-of-list-simps(2) prod.collapse)
    also from Cons.prems(1,2) and Cons.IH have ... = (fmap-of-list ps)(k $$:= v)(fst p $$:= snd
p)
      by (fastforce simp add: member-rec(1))
    finally show ?thesis
    proof −
      assume ∗: fmap-of-list (insort-key fst (k, v) (p # ps)) =
        (fmap-of-list ps)(k $$:= v)(fst p $$:= snd p)
      from Cons.prems(2) have k ∉ set (fst p # map fst ps)
        by simp
      then have ∗∗: {k $$:= v} $$ (fst p) = None
        by (fastforce simp add: member-rec(1))
      have fmap-of-list (p # ps) = (fmap-of-list ps)(fst p $$:= snd p)
        by (metis fmap-of-list-simps(2) prod.collapse)
      with ∗ and ∗∗ show ?thesis
        using fmap-singleton-comm by (metis fmadd-fmupd fmap-of-list-simps(1,2) fmupd-alt-def)
    qed
  qed
qed

lemma fmap-of-list-insort-key-fst-map:
  assumes inj f
  and m $$ k = None
  shows fmap-of-list (insort-key fst (f k, v) (map (λ(k, v). (f k, v)) (sorted-list-of-fmap m))) =
```

```
    (fmap-of-list (map (λ(k, v). (f k, v)) (sorted-list-of-fmap m)))(f k $$:= v)
proof −
  let ?g = λ(k, v). (f k, v)
  let ?ps = map ?g (sorted-list-of-fmap m)
  from ‹inj f› have distinct (map fst ?ps)
    using distinct-fst-inj and distinct-sorted-list-of-fmap by fastforce
  moreover have f k ∉ set (map fst ?ps)
  proof −
    from ‹m $$ k = None› have k ∉ set (map fst (sorted-list-of-fmap m))
      by (metis map-of-eq-None-iff map-of-sorted-list set-map)
    with ‹inj f› show ?thesis
      using map-inj-pair-non-membership by force
  qed
  ultimately show ?thesis
    using fmap-of-list-insort-key-fst by fast
qed

lemma fmap-of-list-sorted-list-of-fmap:
  fixes m :: ('a::linorder, 'b) fmap
  and f :: 'a ⇒ 'c::linorder
  assumes inj f
  and mono f
  and m $$ k = None
  shows fmap-of-list (map (λ(k, v). (f k, v)) (sorted-list-of-fmap (m(k $$:= v)))) =
    (fmap-of-list (map (λ(k, v). (f k, v)) (sorted-list-of-fmap m)))(f k $$:= v)
proof −
  let ?g = λ(k, v). (f k, v)
  from assms(3) have fmap-of-list (map ?g (sorted-list-of-fmap (m(k $$:= v)))) =
    fmap-of-list (map ?g (insort-key fst (k, v) (sorted-list-of-fmap m)))
    by (simp add: sorted-list-of-fmap-is-insort-key-fst)
  also from assms have ... = fmap-of-list (insort-key fst (f k, v) (map ?g (sorted-list-of-fmap m)))
    using calculation and map-sorted-list-of-fmap by fastforce
  also from assms(1,3) have ... = (fmap-of-list (map ?g (sorted-list-of-fmap m)))(f k $$:= v)
    by (simp add: fmap-of-list-insort-key-fst-map)
  finally show ?thesis .
qed
```

Map difference

```
lemma fsubset-antisym:
  assumes m ⊆f n
  and n ⊆f m
  shows m = n
proof −
  from ‹m ⊆f n› have ∀ k ∈ dom (($$) m). (($$) m) k = (($$) n) k
    by (simp add: fmsubset.rep-eq map-le-def)
  moreover from ‹n ⊆f m› have ∀ k ∈ dom (($$) n). (($$) n) k = (($$) m) k
    by (simp add: fmsubset.rep-eq map-le-def)
  ultimately show ?thesis
  proof (intro fmap-ext)
```

```
    fix k
    consider
      (a) k ∈ dom (($$) m) |
      (b) k ∈ dom (($$) n) |
      (c) k ∉ dom (($$) m) ∧ k ∉ dom (($$) n)
      by auto
    then show m $$ k = n $$ k
    proof cases
      case a
      with ‹∀k ∈ dom (($$) m). m $$ k = n $$ k› show ?thesis
        by simp
    next
      case b
      with ‹∀k ∈ dom (($$) n). n $$ k = m $$ k› show ?thesis
        by simp
    next
      case c
      then show ?thesis
        by (simp add: fmdom'-notD)
    qed
  qed
qed

abbreviation
  fmdiff :: ('a, 'b) fmap ⇒ ('a, 'b) fmap ⇒ ('a, 'b) fmap (infixl ‹−−f› 100) where
  m1 −−f m2 ≡ fmfilter (λx. x ∉ fmdom' m2) m1

lemma fmdiff-partition:
  assumes m2 ⊆f m1
  shows m2 ++f (m1 −−f m2) = m1
proof −
  have ∗: m2 ++f (m1 −−f m2) ⊆f m1
  proof −
    have ∀k v. (m2 ++f (m1 −−f m2)) $$ k = Some v ⟶ m1 $$ k = Some v
    proof (intro allI impI)
      fix k v
      assume (m2 ++f (m1 −−f m2)) $$ k = Some v
      then have ∗∗: (if k |∈| fmdom (m1 −−f m2) then (m1 −−f m2) $$ k else m2 $$ k) = Some v
        by simp
      then show m1 $$ k = Some v
      proof (cases k |∈| fmdom (m1 −−f m2))
        case True
        with ∗∗ show ?thesis
          by simp
      next
        case False
        with ∗∗ and ‹m2 ⊆f m1› show ?thesis
          by (metis (mono-tags, lifting) fmpredD fmsubset-alt-def)
      qed
```

```
    qed
    then have fmpred (λk v. m₁ $$ k = Some v) (m₂ ++ₓ (m₁ --ₓ m₂))
      by (blast intro: fmpred-iff)
    then show ?thesis
      by (auto simp add: fmsubset-alt-def)
  qed
  then have m₁ ⊆ₓ m₂ ++ₓ (m₁ --ₓ m₂)
    by (simp add: fmsubset.rep-eq map-le-def)
  with ∗ show ?thesis
    by (simp add: fsubset-antisym)
qed

lemma fmdiff-fmupd:
  assumes m $$ k = None
  shows m(k $$:= v) --ₓ {k $$:= v} = m
proof −
  let ?P = (λk'. k' ∉ {k})
  have m(k $$:= v) --ₓ {k $$:= v} = fmfilter (λx. x ∉ fmdom' {k $$:= v}) (m(k $$:= v)) ..
  also have ... = fmfilter ?P (m(k $$:= v))
    by simp
  also have ... = (if ?P k then (fmfilter ?P m)(k $$:= v) else fmfilter ?P m)
    by simp
  also have ... = fmfilter ?P m
    by simp
  finally show ?thesis
  proof −
    from ‹m $$ k = None› have ⋀k' v'. m $$ k' = Some v' ⟹ ?P k'
      by fastforce
    then show ?thesis
      by simp
  qed
qed
```

Map symmetric difference

```
abbreviation fmsym-diff :: ('a, 'b) fmap ⇒ ('a, 'b) fmap ⇒ ('a, 'b) fmap (infixl ‹Δₓ› 100) where
  m₁ Δₓ m₂ ≡ (m₁ --ₓ m₂) ++ₓ (m₂ --ₓ m₁)
```

Domain restriction

```
abbreviation dom-res :: 'a set ⇒ ('a, 'b) fmap ⇒ ('a, 'b) fmap (infixl ‹◁› 150) where
  s ◁ m ≡ fmfilter (λx. x ∈ s) m
```

Domain exclusion

```
abbreviation dom-exc :: 'a set ⇒ ('a, 'b) fmap ⇒ ('a, 'b) fmap (infixl ‹◁'/› 150) where
  s ◁/ m ≡ fmfilter (λx. x ∉ s) m
```

Intersection plus

```
abbreviation intersection-plus :: ('a, 'b::monoid-add) fmap ⇒ ('a, 'b) fmap ⇒ ('a, 'b) fmap
  (infixl ‹∩₊› 100)
```

where
$m_1 \cap_+ m_2 \equiv$ *fmmap-keys* $(\lambda k\ v.\ v + m_1\ \$\$!\ k)$ $(fmdom'\ m_1 \lhd m_2)$

Union override right

abbreviation *union-override-right* :: $('a, 'b)$ *fmap* $\Rightarrow$ $('a, 'b)$ *fmap* $\Rightarrow$ $('a, 'b)$ *fmap*
(**infixl** ‹$\cup_\rightarrow$› *100*)
where
$m_1 \cup_\rightarrow m_2 \equiv (fmdom'\ m_2 \lhd/\ m_1) +\!\!+_f m_2$

Union override left

abbreviation *union-override-left* :: $('a, 'b)$ *fmap* $\Rightarrow$ $('a, 'b)$ *fmap* $\Rightarrow$ $('a, 'b)$ *fmap*
(**infixl** ‹$\cup_\leftarrow$› *100*)
where
$m_1 \cup_\leftarrow m_2 \equiv m_1 +\!\!+_f (fmdom'\ m_1 \lhd/\ m_2)$

Union override plus

abbreviation *union-override-plus* :: $('a, 'b$::*monoid-add*$)$ *fmap* $\Rightarrow$ $('a, 'b)$ *fmap* $\Rightarrow$ $('a, 'b)$ *fmap*
(**infixl** ‹$\cup_+$› *100*)
where
$m_1 \cup_+ m_2 \equiv (m_1\ \Delta_f\ m_2) +\!\!+_f (m_1 \cap_+ m_2)$

Extra lemmas for the non-standard map operators

lemma *dom-res-singleton*:
assumes $m\ \$\$\ k = Some\ v$
shows $\{k\} \lhd m = \{k\ \$\$:= v\}$
using *assms*
proof (*induction m*)
case *fmempty*
then show *?case*
by *simp*
next
case (*fmupd k' v' m*)
then show *?case*
proof (*cases k = k'*)
case *True*
with ‹$m(k'\ \$\$:= v')\ \$\$\ k = Some\ v$› **have** $v = v'$
by *simp*
with *True* **have** $\{k\} \lhd m(k'\ \$\$:= v') = (\{k\} \lhd m)(k\ \$\$:= v)$
by *simp*
also from *True* **and** ‹$m\ \$\$\ k' = None$› **have** $\ldots = \{\$\$\}(k\ \$\$:= v)$
by (*simp add*: *fmap-ext*)
finally show *?thesis*
by *simp*
next
case *False*
with ‹$m(k'\ \$\$:= v')\ \$\$\ k = Some\ v$› **have** $*$: $m\ \$\$\ k = Some\ v$
by *simp*
with *False* **have** $\{k\} \lhd m(k'\ \$\$:= v') = \{k\} \lhd m$

```
    by simp
  with * and fmupd.IH show ?thesis
    by simp
 qed
qed

lemma dom-res-union-distr:
 shows (A ∪ B) ⊲ m = A ⊲ m ++_f B ⊲ m
proof −
 have ($$) ((A ∪ B) ⊲ m) ⊆_m ($$) (A ⊲ m ++_f B ⊲ m)
 proof (unfold map-le-def, intro ballI)
  fix k
  assume k ∈ dom (($$) ((A ∪ B) ⊲ m))
  then show ((A ∪ B) ⊲ m) $$ k = (A ⊲ m ++_f B ⊲ m) $$ k
   by auto
 qed
 moreover have ($$) (A ⊲ m ++_f B ⊲ m) ⊆_m ($$) ((A ∪ B) ⊲ m)
 proof (unfold map-le-def, intro ballI)
  fix k
  assume k ∈ dom (($$) (A ⊲ m ++_f B ⊲ m))
  then show (A ⊲ m ++_f B ⊲ m) $$ k = ((A ∪ B) ⊲ m) $$ k
   by auto
 qed
 ultimately show ?thesis
  using fmlookup-inject and map-le-antisym by blast
qed

lemma dom-exc-add-distr:
 shows s ⊲/ (m_1 ++_f m_2) = (s ⊲/ m_1) ++_f (s ⊲/ m_2)
 by (blast intro: fmfilter-add-distrib)

lemma fmap-partition:
 shows m = s ⊲/ m ++_f s ⊲ m
proof (induction m)
 case fmempty
 then show ?case
  by simp
next
 case (fmupd k v m)
 from fmupd.hyps have s ⊲/ m(k $$:= v) ++_f s ⊲ m(k $$:= v) =
  s ⊲/ (m ++_f {k $$:= v}) ++_f s ⊲ (m ++_f {k $$:= v})
  by simp
 also have ... = s ⊲/ m ++_f s ⊲/ {k $$:= v} ++_f s ⊲ m ++_f s ⊲ {k $$:= v}
  using dom-exc-add-distr by simp
 finally show ?case
 proof (cases k ∈ s)
  case True
  then have s ⊲/ m ++_f s ⊲/ {k $$:= v} ++_f s ⊲ m ++_f s ⊲ {k $$:= v} =
   s ⊲/ m ++_f {$$} ++_f s ⊲ m ++_f {k $$:= v}
```

```
    by simp
  also have ... = s ⩤/ m ++_f s ⩤ m ++_f {k $$:= v}
    by simp
  also from fmupd.IH have ... = m ++_f {k $$:= v}
    by simp
  finally show ?thesis using fmupd.hyps
    by auto
next
  case False
  then have s ⩤/ m ++_f s ⩤/ {k $$:= v} ++_f s ⩤ m ++_f s ⩤ {k $$:= v} =
    s ⩤/ m ++_f {k $$:= v} ++_f s ⩤ m ++_f {$$}
    by simp
  also have ... = s ⩤/ m ++_f {k $$:= v} ++_f s ⩤ m
    by simp
  also from fmupd.hyps have ... = s ⩤/ m ++_f s ⩤ m ++_f {k $$:= v}
    using fmap-singleton-comm by (metis (full-types) fmadd-assoc fmlookup-filter)
  also from fmupd.IH have ... = m ++_f {k $$:= v}
    by simp
  finally show ?thesis
    by auto
qed
qed

lemma dom-res-addition-in:
  assumes m1 $$ k = None
  and m2 $$ k = Some v'
  shows fmdom' (m1(k $$:= v)) ⩤ m2 = fmdom' m1 ⩤ m2 ++_f {k $$:= v'}
proof −
  from ‹m1 $$ k = None› have fmdom' (m1(k $$:= v)) ⩤ m2 = (fmdom' m1 ∪ {k}) ⩤ m2
    by simp
  also have ... = fmdom' m1 ⩤ m2 ++_f {k} ⩤ m2
    using dom-res-union-distr .
  finally show ?thesis
    using ‹m2 $$ k = Some v'› and dom-res-singleton by fastforce
qed

lemma dom-res-addition-not-in:
  assumes m2 $$ k = None
  shows fmdom' (m1(k $$:= v)) ⩤ m2 = fmdom' m1 ⩤ m2
proof −
  have fmdom' (m1(k $$:= v)) ⩤ m2 = fmfilter (λk'. k' = k ∨ k' ∈ fmdom' m1) m2
    by simp
  also have ... = fmdom' m1 ⩤ m2
  proof (intro fmfilter-cong')
    show m2 = m2 ..
  next
    from assms show k' ∈ fmdom' m2 ⟹ (k' = k ∨ k' ∈ fmdom' m1) = (k' ∈ fmdom' m1) for k'
    by (cases k' = k) (simp-all add: fmdom'-notI)
  qed
```

finally show *?thesis* **.**
qed

lemma *inter-plus-addition-in*:
assumes m_1 $\$\$\ k = None$
and m_2 $\$\$\ k = Some\ v'$
shows $m_1(k\ \$\$:= v) \cap_+ m_2 = (m_1 \cap_+ m_2) +\!\!+_f \{k\ \$\$:= v' + v\}$
proof −
let *?f* $= \lambda k'\ v'.\ v' + m_1(k\ \$\$:= v)\ \$\$!\ k'$
from *assms* **have** $m_1(k\ \$\$:= v) \cap_+ m_2 =$ *fmmap-keys ?f* $((fmdom'\ m_1 \lhd m_2) +\!\!+_f \{k\ \$\$:= v'\})$
using *dom-res-addition-in* **by** *fastforce*
also have $\ldots =$ *fmmap-keys ?f* $(fmdom'\ m_1 \lhd m_2) +\!\!+_f$ *fmmap-keys ?f* $\{k\ \$\$:= v'\}$
proof −
from ‹m_1 $\$\$\ k = None$› **have** $fmdom'\ (fmdom'\ m_1 \lhd m_2) \cap fmdom'\ \{k\ \$\$:= v'\} = \{\}$
by (*simp add*: *fmdom'-notI*)
then show *?thesis*
using *fmmap-keys-hom* **by** *blast*
qed
also from *assms*
have $\ldots =$ *fmmap-keys ?f* $(fmdom'\ m_1 \lhd m_2) +\!\!+_f \{k\ \$\$:= v' + v\}$
proof −
have *fmmap-keys ?f* $\{k\ \$\$:= v'\} = \{k\ \$\$:= v' + v\}$
proof (*intro fmap-ext*)
fix k'
have *fmmap-keys ?f* $\{k\ \$\$:= v'\}\ \$\$\ k' =$ *map-option* $(?f\ k')$ $(\{k\ \$\$:= v'\}\ \$\$\ k')$
using *fmlookup-fmmap-keys* **.**
also have $\ldots =$ *map-option* $(?f\ k')$ $(if\ k = k'\ then\ Some\ v'\ else\ \{\$\$\}\ \$\$\ k')$
by *simp*
also have $\ldots = \{k\ \$\$:= v' + v\}\ \$\$\ k'$
by (*cases* $k' = k$) *simp-all*
finally show *fmmap-keys ?f* $\{k\ \$\$:= v'\}\ \$\$\ k' = \{k\ \$\$:= v' + v\}\ \$\$\ k'$ **.**
qed
then show *?thesis*
by *simp*
qed
also have $\ldots =$ *fmmap-keys* $(\lambda k'\ v'.\ v' + m_1\ \$\$!\ k')$ $(fmdom'\ m_1 \lhd m_2) +\!\!+_f \{k\ \$\$:= v' + v\}$
by (*simp add*: *fmap-ext*)
finally show *?thesis* **.**
qed

lemma *inter-plus-addition-notin*:
assumes m_1 $\$\$\ k = None$
and m_2 $\$\$\ k = None$
shows $m_1(k\ \$\$:= v) \cap_+ m_2 = (m_1 \cap_+ m_2)$
proof −
let *?f* $= \lambda k'\ v'.\ v' + m_1(k\ \$\$:= v)\ \$\$!\ k'$
from ‹m_2 $\$\$\ k = None$›
have $m_1(k\ \$\$:= v) \cap_+ m_2 =$ *fmmap-keys ?f* $(fmdom'\ m_1 \lhd m_2)$
using *dom-res-addition-not-in* **by** *metis*

```
  also have ... = fmmap-keys (λk' v'. v' + m1 $$! k') (fmdom' m1 ◁ m2)
  proof (intro fmap-ext)
    fix k'
    have fmmap-keys ?f (fmdom' m1 ◁ m2) $$ k' = map-option (?f k') ((fmdom' m1 ◁ m2) $$ k')
      using fmlookup-fmmap-keys .
    also from ‹m1 $$ k = None› have ... = fmmap-keys (λk' v'. v' + m1 $$! k') (fmdom' m1 ◁ m2)
$$ k'
      by (cases k' = k) (simp-all add: fmdom'-notI)
    finally show fmmap-keys ?f (fmdom' m1 ◁ m2) $$ k' =
      fmmap-keys (λk' v'. v' + m1 $$! k') (fmdom' m1 ◁ m2) $$ k' .
  qed
  finally show ?thesis .
qed

lemma union-plus-addition-notin:
  assumes m1 $$ k = None
  and m2 $$ k = None
  shows m1(k $$:= v) ∪+ m2 = (m1 ∪+ m2) ++f {k $$:= v}
proof −
  from ‹m2 $$ k = None› have (m1(k $$:= v)) ∪+ m2 =
    fmdom' m2 ◁/ m1 ++f {k $$:= v} ++f fmdom' (m1(k $$:= v)) ◁/ m2 ++f (m1(k $$:= v) ∩+
m2)
    by (simp add: fmdom'-notI)
  also from assms have ... =
    fmdom' m2 ◁/ m1 ++f {k $$:= v} ++f fmdom' m1 ◁/ m2 ++f (m1(k $$:= v) ∩+ m2)
    by (smt fmdom'-fmupd fmfilter-cong insert-iff option.distinct(1))
  also from assms have ... = fmdom' m2 ◁/ m1 ++f {k $$:= v} ++f fmdom' m1 ◁/ m2 ++f
(m1 ∩+ m2)
    using inter-plus-addition-notin by metis
  also from assms have ... = fmdom' m2 ◁/ m1 ++f fmdom' m1 ◁/ m2 ++f (m1 ∩+ m2) ++f
{k $$:= v}
    using fmap-singleton-comm
    by (smt fmadd-assoc fmfilter-fmmap-keys fmlookup-filter fmlookup-fmmap-keys)
  finally show ?thesis .
qed

end
```