

# Finite Map Extras

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

May 26, 2024

## 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**  $\langle\$\$\rangle$  900)

**notation** *fmempty* ( $\langle\{\$\$\}\rangle$ )

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

**syntax**

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

**translations**

$-FmapUpd m (-Fmaplets xy ms) \equiv -FmapUpd (-FmapUpd m xy) ms$   
 $-FmapUpd m (-fmaplet x y) \equiv CONST fmupd x y m$   
 $-Fmap ms \equiv -FmapUpd (CONST fmempty) ms$   
 $-Fmap (-Fmaplets ms1 ms2) \leftarrow -FmapUpd (-Fmap ms1) ms2$

$-Fmaplets\ ms1\ (-Fmaplets\ ms2\ ms3) \leftarrow -Fmaplets\ (-Fmaplets\ ms1\ ms2)\ ms3$

**abbreviation** *fmap-lookup-the* (**infixl**  $\langle \$\$! \rangle\ 900$ ) **where**  
 $m\ \$\$!\ k \equiv the\ (m\ \$\$ k)$

**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)  $k = k_1 \mid$

(b)  $k = k_2 \mid$

(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**  $\dots = \{x\ \$\$:= y\} ++_f m ++_f \{k\ \$\$:= v\}$

**by** *simp*

**also from** *fmupd.prem*s **and** *fmupd.hyps* **and** *fmupd.IH* **have**  $\dots = \{x\ \$\$:= y\} ++_f \{k\ \$\$:= v\} ++_f m$

**by** (*metis fmadd-assoc fmupd-lookup*)

**also have**  $\dots = \{k\ \$\$:= v\} ++_f m(x\ \$\$:= y)$

**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.prem*s **and** *fmupd.hyps* **and** *fmupd.IH* **have**  $\dots = \{k\ \$\$:= v\} ++_f m ++_f \{x\ \$\$:= y\}$

**by** (*metis fmadd-assoc*)

**finally show** *?thesis*

**by** *simp*

**next**

**case** *False*

**with** *fmupd.prem*s **show** *?thesis*

```

    by auto
  qed
  finally show ?case .
qed

```

lemma *fmap-disj-comm*:

```

  assumes  $fmdom' m_1 \cap fmdom' m_2 = \{\}$ 
  shows  $m_1 ++_f m_2 = m_2 ++_f m_1$ 
  using assms
proof (induction  $m_2$  arbitrary:  $m_1$ )
  case fmempty
  then show ?case
    by simp
next
  case (fmupd  $k v m_2$ )
  then show ?case
  proof (cases  $m_1 \ \$\$ k = None$ )
    case True
    from fmupd.hyps have  $m_1 ++_f m_2(k \ \$\$ := v) = m_1 ++_f m_2 ++_f \{k \ \$\$ := v\}$ 
      by simp
    also from fmupd.prems and fmupd.hyps and fmupd.IH have  $\dots = m_2 ++_f m_1 ++_f \{k \ \$\$ := v\}$ 
      by simp
    also from True have  $\dots = m_2 ++_f \{k \ \$\$ := v\} ++_f m_1$ 
      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  $v' \in | \ fmran \{k \ \$\$ := v\} \implies v' = v$  for  $v'$ 
  proof -
    assume  $v' \in | \ fmran \{k \ \$\$ := v\}$ 
    fix  $k'$ 
    have  $fmdom' \{k \ \$\$ := v\} = \{k\}$ 
      by simp
    then show  $v' = v$ 
  proof (cases  $k' = k$ )
    case True
    with  $\langle v' \in | \ fmran \{k \ \$\$ := v\} \rangle$  show ?thesis
      using fmdom'I by fastforce
  next
    case False
    with  $\langle fmdom' \{k \ \$\$ := v\} = \{k\} \rangle$  and  $\langle v' \in | \ fmran \{k \ \$\$ := v\} \rangle$  show ?thesis

```

using *fmdom'I* by *fastforce*  
 qed  
 qed  
 moreover have  $v \in \text{fmdom } \{k \text{ $$$} := v\}$   
 by (*simp add: fmdomI*)  
 ultimately show *?thesis*  
 by (*simp add: fsubsetI fsubset-antisym*)  
 qed

**lemma** *fmap-keys-hom*:  
 assumes  $\text{fmdom}' m_1 \cap \text{fmdom}' m_2 = \{\}$   
 shows  $\text{fmap-keys } f m_1 ++_f \text{fmap-keys } f m_2 = \text{fmap-keys } f m_1 ++_f \text{fmap-keys } f m_2$   
 using *assms*  
 by (*simp add: fmap-ext*)

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

**lemma** *sorted-list-of-fmap-is-insort-key-fst*:  
 assumes  $m \text{ $$$ } k = \text{None}$   
 shows  $\text{sorted-list-of-fmap } (m(k \text{ $$$} := v)) = \text{insort-key } fst (k, v) (\text{sorted-list-of-fmap } m)$

**proof** –  
 have  $\text{sorted-list-of-fmap } (m(k \text{ $$$} := v)) =$   
 $\text{map } (\lambda k'. (k', m(k \text{ $$$} := v) \text{ $$$! } k')) (\text{sorted-list-of-fset } (\text{fmdom } (m(k \text{ $$$} := v))))$   
 unfolding *sorted-list-of-fmap-def* ..  
 also have  $\dots = \text{map } (\lambda k'. (k', m(k \text{ $$$} := v) \text{ $$$! } k')) (\text{sorted-list-of-fset } (\text{finsert } k (\text{fmdom } m)))$   
 by *simp*  
 also from  $\langle m \text{ $$$ } k = \text{None} \rangle$  have  $\dots =$   
 $\text{map } (\lambda k'. (k', m(k \text{ $$$} := v) \text{ $$$! } k')) (\text{insort } k (\text{sorted-list-of-fset } (\text{fmdom } m - \{k\})))$   
 by (*simp add: sorted-list-of-fset.rep-eq*)  
 also from  $\langle m \text{ $$$ } k = \text{None} \rangle$  have  $\dots =$   
 $\text{map } (\lambda k'. (k', m(k \text{ $$$} := v) \text{ $$$! } k')) (\text{insort } k (\text{sorted-list-of-fset } (\text{fmdom } m)))$   
 by (*simp add: fmdom-notI*)  
 also from  $\langle m \text{ $$$ } k = \text{None} \rangle$  have  $\dots =$   
 $\text{insort-key } fst (k, v) (\text{map } (\lambda k'. (k', m(k \text{ $$$} := v) \text{ $$$! } k')) (\text{sorted-list-of-fset } (\text{fmdom } m)))$   
 using *map-insort-is-insort-key* by *fastforce*  
 also have  $\dots = \text{insort-key } fst (k, v) (\text{map } (\lambda k'. (k', m \text{ $$$! } k')) (\text{sorted-list-of-fset } (\text{fmdom } m)))$   
**proof** –  
 from  $\langle m \text{ $$$ } k = \text{None} \rangle$  have  $\bigwedge k'. k' \in \text{fmdom}' m \implies m(k \text{ $$$} := v) \text{ $$$! } k' = m \text{ $$$! } k'$   
 using *fmdom'-notI* by *force*  
 moreover from  $\langle m \text{ $$$ } k = \text{None} \rangle$  have  $k \notin \text{set } (\text{sorted-list-of-fset } (\text{fmdom } m))$   
 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 ( $\lambda(k, v). (f k, v)$ ) ps))*  
**proof** –  
  **have** *map fst (map ( $\lambda(k, v). (f k, v)$ ) 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  $\notin$  set (map fst ps)*  
  **and** *inj f*  
  **shows** *f k  $\notin$  set (map fst (map ( $\lambda(k, v). (f k, v)$ ) 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  $\notin$  set (map fst ps)*  
  **and** *inj f*  
  **and** *mono f*  
  **shows** *map ( $\lambda(k, v). (f k, v)$ ) (insort-key fst (k, v) ps) =*  
    *insort-key fst (f k, v) (map ( $\lambda(k, v). (f k, v)$ ) ps)*  
  **using** *assms*  
**proof** (*induction ps*)  
  **case** *Nil*  
  **then show** *?case*  
    **by** *simp*  
**next**  
  **let** *?g = ( $\lambda(k, v). (f k, v)$ )*  
  **case** (*Cons p ps*)  
  **then show** *?case*  
  **proof** (*cases k  $\leq$  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.prem(4)* **and** *True* **have** *f k  $\leq$  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.prem1,2* **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  $k \notin \text{set } (\text{map fst } ps)$

shows  $\text{fmap-of-list } (\text{insort-key fst } (k, v) ps) = (\text{fmap-of-list } ps)(k \text{ \&\&:= } v)$

using *assms*

proof (induction ps)

case Nil

then show ?case

by *simp*

next

case (Cons p ps)

then show ?case

proof (cases  $k \leq \text{fst } p$ )

case True

then show ?thesis

by *simp*

next

case False

then have  $\text{fmap-of-list } (\text{insort-key fst } (k, v) (p \# ps)) =$

$\text{fmap-of-list } (p \# \text{insort-key fst } (k, v) ps)$

by *simp*

also have  $\dots = (\text{fmap-of-list } (\text{insort-key fst } (k, v) ps))(\text{fst } p \text{ \&\&:= } \text{snd } p)$

by (metis *fmap-of-list-simps(2) prod.collapse*)

also from *Cons.prem(1,2)* and *Cons.IH* have  $\dots = (\text{fmap-of-list } ps)(k \text{ \&\&:= } v)(\text{fst } p \text{ \&\&:= } \text{snd}$

$p)$

by (fastforce *simp add: member-rec(1)*)

finally show ?thesis

proof –

assume \*:  $\text{fmap-of-list } (\text{insort-key fst } (k, v) (p \# ps)) =$

$(\text{fmap-of-list } ps)(k \text{ \&\&:= } v)(\text{fst } p \text{ \&\&:= } \text{snd } p)$

from *Cons.prem(2)* have  $k \notin \text{set } (\text{fst } p \# \text{map fst } ps)$

by *simp*

then have \*\*:  $\{k \text{ \&\&:= } v\} \text{ \&\& } (\text{fst } p) = \text{None}$

by (fastforce *simp add: member-rec(1)*)

have  $\text{fmap-of-list } (p \# ps) = (\text{fmap-of-list } ps)(\text{fst } p \text{ \&\&:= } \text{snd } p)$

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  $m \text{ \&\& } k = \text{None}$

shows  $\text{fmap-of-list } (\text{insort-key fst } (f k, v) (\text{map } (\lambda(k, v). (f k, v)) (\text{sorted-list-of-fmap } m))) =$

$(fmap\text{-of-list } (map (\lambda(k, v). (f k, v)) (sorted\text{-list-of-fmap } m)))(f k \text{ \&S;:= } v)$   
**proof** –  
**let**  $?g = \lambda(k, v). (f k, v)$   
**let**  $?ps = map ?g (sorted\text{-list-of-fmap } m)$   
**from**  $\langle inj f \rangle$  **have**  $distinct (map fst ?ps)$   
**using**  $distinct\text{-fst-inj}$  **and**  $distinct\text{-sorted-list-of-fmap}$  **by**  $fastforce$   
**moreover** **have**  $f k \notin set (map fst ?ps)$   
**proof** –  
**from**  $\langle m \text{ \&S; } k = None \rangle$  **have**  $k \notin set (map fst (sorted\text{-list-of-fmap } m))$   
**by**  $(metis map\text{-of-eq-None-iff map\text{-of-sorted-list set-map})$   
**with**  $\langle inj f \rangle$  **show**  $?thesis$   
**using**  $map\text{-inj-pair-non-membership}$  **by**  $force$   
**qed**  
**ultimately show**  $?thesis$   
**using**  $fmap\text{-of-list-insort-key-fst}$  **by**  $fast$   
**qed**

**lemma**  $fmap\text{-of-list-sorted-list-of-fmap}$ :

**fixes**  $m :: ('a::linorder, 'b) fmap$   
**and**  $f :: 'a \Rightarrow 'c::linorder$   
**assumes**  $inj f$   
**and**  $mono f$   
**and**  $m \text{ \&S; } k = None$   
**shows**  $fmap\text{-of-list } (map (\lambda(k, v). (f k, v)) (sorted\text{-list-of-fmap } (m(k \text{ \&S;:= } v)))) =$   
 $(fmap\text{-of-list } (map (\lambda(k, v). (f k, v)) (sorted\text{-list-of-fmap } m)))(f k \text{ \&S;:= } v)$

**proof** –  
**let**  $?g = \lambda(k, v). (f k, v)$   
**from**  $assms(3)$  **have**  $fmap\text{-of-list } (map ?g (sorted\text{-list-of-fmap } (m(k \text{ \&S;:= } v)))) =$   
 $fmap\text{-of-list } (map ?g (insort\text{-key } fst (k, v) (sorted\text{-list-of-fmap } m)))$   
**by**  $(simp add: sorted\text{-list-of-fmap-is-insort-key-fst})$   
**also from**  $assms$  **have**  $\dots = fmap\text{-of-list } (insort\text{-key } fst (f k, v) (map ?g (sorted\text{-list-of-fmap } m)))$   
**using**  $calculation$  **and**  $map\text{-sorted-list-of-fmap}$  **by**  $fastforce$   
**also from**  $assms(1,3)$  **have**  $\dots = (fmap\text{-of-list } (map ?g (sorted\text{-list-of-fmap } m)))(f k \text{ \&S;:= } v)$   
**by**  $(simp add: fmap\text{-of-list-insort-key-fst-map})$   
**finally show**  $?thesis$  .  
**qed**

Map difference

**lemma**  $fsubset\text{-antisym}$ :

**assumes**  $m \subseteq_f n$   
**and**  $n \subseteq_f m$   
**shows**  $m = n$   
**proof** –  
**from**  $\langle m \subseteq_f n \rangle$  **have**  $\forall k \in dom ((\text{\&S;}) m). ((\text{\&S;}) m) k = ((\text{\&S;}) n) k$   
**by**  $(simp add: fmsubset.rep-eq map-le-def)$   
**moreover from**  $\langle n \subseteq_f m \rangle$  **have**  $\forall k \in dom ((\text{\&S;}) n). ((\text{\&S;}) n) k = ((\text{\&S;}) m) k$   
**by**  $(simp add: fmsubset.rep-eq map-le-def)$   
**ultimately show**  $?thesis$   
**proof**  $(intro fmap-ext)$



```

fix k
consider
  (a)  $k \in \text{dom } ((\$\$) m) \mid$ 
  (b)  $k \in \text{dom } ((\$\$) n) \mid$ 
  (c)  $k \notin \text{dom } ((\$\$) m) \wedge k \notin \text{dom } ((\$\$) n)$ 
  by auto
then show  $m \ \$\$ k = n \ \$\$ k$ 
proof cases
  case a
  with  $\langle \forall k \in \text{dom } ((\$\$) m). m \ \$\$ k = n \ \$\$ k \rangle$  show ?thesis
  by simp
next
  case b
  with  $\langle \forall k \in \text{dom } ((\$\$) n). n \ \$\$ k = m \ \$\$ k \rangle$  show ?thesis
  by simp
next
  case c
  then show ?thesis
  by (simp add: fmdom'-notD)
qed
qed
qed

```

#### abbreviation

$fmdiff :: ('a, 'b) fmap \Rightarrow ('a, 'b) fmap \Rightarrow ('a, 'b) fmap$  (**infixl**  $\langle \_ \_ \_ \rangle$  100) **where**  
 $m_1 \_ \_ \_ m_2 \equiv fmfiltter (\lambda x. x \notin fmdom' m_2) m_1$

#### lemma *fmdiff-partition*:

```

assumes  $m_2 \subseteq_f m_1$ 
shows  $m_2 \ ++_f (m_1 \_ \_ \_ m_2) = m_1$ 
proof  $-$ 
  have  $*$ :  $m_2 \ ++_f (m_1 \_ \_ \_ m_2) \subseteq_f m_1$ 
  proof  $-$ 
    have  $\forall k v. (m_2 \ ++_f (m_1 \_ \_ \_ m_2)) \ \$\$ k = Some v \longrightarrow m_1 \ \$\$ k = Some v$ 
    proof (intro allI impI)
      fix k v
      assume  $(m_2 \ ++_f (m_1 \_ \_ \_ m_2)) \ \$\$ k = Some v$ 
      then have  $**$ : (if  $k \in | fmdom (m_1 \_ \_ \_ m_2)$  then  $(m_1 \_ \_ \_ m_2) \ \$\$ k$  else  $m_2 \ \$\$ k = Some v$ )
      by simp
      then show  $m_1 \ \$\$ k = Some v$ 
    proof (cases k |∈| fmdom (m_1 \_ \_ \_ m_2))
      case True
      with  $**$  show ?thesis
      by simp
    next
      case False
      with  $**$  and  $\langle m_2 \subseteq_f m_1 \rangle$  show ?thesis
      by (metis (mono-tags, lifting) fmpredD fmsubset-alt-def)
    qed
  qed

```

**qed**  
**then have**  $fmpred (\lambda k v. m_1 \text{ \&\& } k = \text{Some } v) (m_2 ++_f (m_1 --_f m_2))$   
**by** (*blast intro: fmpred-iff*)  
**then show** *?thesis*  
**by** (*auto simp add: fmsubset-alt-def*)  
**qed**  
**then have**  $m_1 \subseteq_f m_2 ++_f (m_1 --_f m_2)$   
**by** (*simp add: fmsubset.rep-eq map-le-def*)  
**with \* show** *?thesis*  
**by** (*simp add: fsubset-antisym*)  
**qed**

**lemma** *fmdiff-fmupd:*

**assumes**  $m \text{ \&\& } k = \text{None}$   
**shows**  $m(k \text{ \&\& } = v) --_f \{k \text{ \&\& } = v\} = m$   
**proof** –  
**let**  $?P = (\lambda k'. k' \notin \{k\})$   
**have**  $m(k \text{ \&\& } = v) --_f \{k \text{ \&\& } = v\} = fmfiltter (\lambda x. x \notin fmdom' \{k \text{ \&\& } = v\}) (m(k \text{ \&\& } = v)) \dots$   
**also have**  $\dots = fmfiltter ?P (m(k \text{ \&\& } = v))$   
**by** *simp*  
**also have**  $\dots = (\text{if } ?P k \text{ then } (fmfiltter ?P m)(k \text{ \&\& } = v) \text{ else } fmfiltter ?P m)$   
**by** *simp*  
**also have**  $\dots = fmfiltter ?P m$   
**by** *simp*  
**finally show** *?thesis*  
**proof** –  
**from**  $\langle m \text{ \&\& } k = \text{None} \rangle$  **have**  $\bigwedge k' v'. m \text{ \&\& } k' = \text{Some } v' \implies ?P k'$   
**by** *fastforce*  
**then show** *?thesis*  
**by** *simp*  
**qed**  
**qed**

Map symmetric difference

**abbreviation** *fmsym-diff* ::  $('a, 'b) fmap \Rightarrow ('a, 'b) fmap \Rightarrow ('a, 'b) fmap$  (**infixl**  $\langle \Delta_f \rangle$  100) **where**  
 $m_1 \Delta_f m_2 \equiv (m_1 --_f m_2) ++_f (m_2 --_f m_1)$

Domain restriction

**abbreviation** *dom-res* ::  $'a \text{ set} \Rightarrow ('a, 'b) fmap \Rightarrow ('a, 'b) fmap$  (**infixl**  $\langle \triangleleft \rangle$  150) **where**  
 $s \triangleleft m \equiv fmfiltter (\lambda x. x \in s) m$

Domain exclusion

**abbreviation** *dom-exc* ::  $'a \text{ set} \Rightarrow ('a, 'b) fmap \Rightarrow ('a, 'b) fmap$  (**infixl**  $\langle \triangleleft' \rangle$  150) **where**  
 $s \triangleleft' m \equiv fmfiltter (\lambda x. x \notin s) m$

Intersection plus

**abbreviation** *intersection-plus* ::  $('a, 'b) \text{ monoid-add} \Rightarrow ('a, 'b) fmap \Rightarrow ('a, 'b) fmap$   
(**infixl**  $\langle \cap_+ \rangle$  100)

**where**

$$m_1 \cap_+ m_2 \equiv \text{fmap-keys } (\lambda k v. v + m_1 \text{ \&\& } k) (\text{fmdom}' m_1 \triangleleft m_2)$$

Union override right

**abbreviation** *union-override-right* :: ('a, 'b) fmap  $\Rightarrow$  ('a, 'b) fmap  $\Rightarrow$  ('a, 'b) fmap  
(**infixl**  $\langle \cup_{\rightarrow} \rangle$  100)

**where**

$$m_1 \cup_{\rightarrow} m_2 \equiv (\text{fmdom}' m_2 \triangleleft / m_1) ++_f m_2$$

Union override left

**abbreviation** *union-override-left* :: ('a, 'b) fmap  $\Rightarrow$  ('a, 'b) fmap  $\Rightarrow$  ('a, 'b) fmap  
(**infixl**  $\langle \cup_{\leftarrow} \rangle$  100)

**where**

$$m_1 \cup_{\leftarrow} m_2 \equiv m_1 ++_f (\text{fmdom}' m_1 \triangleleft / m_2)$$

Union override plus

**abbreviation** *union-override-plus* :: ('a, 'b::monoid-add) fmap  $\Rightarrow$  ('a, 'b) fmap  $\Rightarrow$  ('a, 'b) fmap  
(**infixl**  $\langle \cup_+ \rangle$  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**  $m \text{ \&\& } k = \text{Some } v$

**shows**  $\{k\} \triangleleft m = \{k \text{ \&\& } = v\}$

**using** *assms*

**proof** (*induction m*)

**case** *fmempty*

**then show** *?case*

**by** *simp*

**next**

**case** (*fmapd k' v' m*)

**then show** *?case*

**proof** (*cases k = k'*)

**case** *True*

**with**  $\langle m(k' \text{ \&\& } = v') \text{ \&\& } k = \text{Some } v \rangle$  **have**  $v = v'$

**by** *simp*

**with** *True* **have**  $\{k\} \triangleleft m(k' \text{ \&\& } = v') = (\{k\} \triangleleft m)(k \text{ \&\& } = v)$

**by** *simp*

**also from** *True* **and**  $\langle m \text{ \&\& } k' = \text{None} \rangle$  **have**  $\dots = \{\text{\&\&}\}(k \text{ \&\& } = v)$

**by** (*simp add: fmap-ext*)

**finally show** *?thesis*

**by** *simp*

**next**

**case** *False*

**with**  $\langle m(k' \text{ \&\& } = v') \text{ \&\& } k = \text{Some } v \rangle$  **have**  $*$ :  $m \text{ \&\& } k = \text{Some } v$

**by** *simp*

**with** *False* **have**  $\{k\} \triangleleft m(k' \text{ \&\& } = v') = \{k\} \triangleleft m$

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

**lemma** *dom-res-union-distr*:

shows  $(A \cup B) \triangleleft m = A \triangleleft m ++_f B \triangleleft m$

**proof** –

have  $(\$\$) ((A \cup B) \triangleleft m) \subseteq_m (\$\$) (A \triangleleft m ++_f B \triangleleft m)$

**proof** (*unfold map-le-def, intro ballI*)

fix *k*

assume  $k \in \text{dom } ((\$\$) ((A \cup B) \triangleleft m))$

then show  $((A \cup B) \triangleleft m) \ \$\$ k = (A \triangleleft m ++_f B \triangleleft m) \ \$\$ k$

by *auto*

qed

moreover have  $(\$\$) (A \triangleleft m ++_f B \triangleleft m) \subseteq_m (\$\$) ((A \cup B) \triangleleft m)$

**proof** (*unfold map-le-def, intro ballI*)

fix *k*

assume  $k \in \text{dom } ((\$\$) (A \triangleleft m ++_f B \triangleleft m))$

then show  $(A \triangleleft m ++_f B \triangleleft m) \ \$\$ k = ((A \cup B) \triangleleft m) \ \$\$ k$

by *auto*

qed

ultimately show *?thesis*

using *fmlookup-inject* and *map-le-antisym* by *blast*

qed

**lemma** *dom-exc-add-distr*:

shows  $s \triangleleft / (m_1 ++_f m_2) = (s \triangleleft / m_1) ++_f (s \triangleleft / m_2)$

by (*blast intro: fmfilter-add-distrib*)

**lemma** *fmap-partition*:

shows  $m = s \triangleleft / m ++_f s \triangleleft m$

**proof** (*induction m*)

case *fmempty*

then show *?case*

by *simp*

next

case (*fmupd k v m*)

from *fmupd.hyps* have  $s \triangleleft / m(k \ \$\$ := v) ++_f s \triangleleft m(k \ \$\$ := v) =$

$s \triangleleft / (m ++_f \{k \ \$\$ := v\}) ++_f s \triangleleft (m ++_f \{k \ \$\$ := v\})$

by *simp*

also have  $\dots = s \triangleleft / m ++_f s \triangleleft / \{k \ \$\$ := v\} ++_f s \triangleleft m ++_f s \triangleleft \{k \ \$\$ := v\}$

using *dom-exc-add-distr* by *simp*

finally show *?case*

**proof** (*cases k \in s*)

case *True*

then have  $s \triangleleft / m ++_f s \triangleleft / \{k \ \$\$ := v\} ++_f s \triangleleft m ++_f s \triangleleft \{k \ \$\$ := v\} =$

$s \triangleleft / m ++_f \{\$\$\} ++_f s \triangleleft m ++_f \{k \ \$\$ := v\}$

```

    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  $m_1 \text{ \&\& } k = \text{None}$

and  $m_2 \text{ \&\& } k = \text{Some } v'$

shows  $m_1(k \text{ \&\& } = v) \cap_+ m_2 = (m_1 \cap_+ m_2) ++_f \{k \text{ \&\& } = v' + v\}$

proof –

let  $?f = \lambda k' v'. v' + m_1(k \text{ \&\& } = v) \text{ \&\& } ! k'$

from *assms* have  $m_1(k \text{ \&\& } = v) \cap_+ m_2 = \text{fmap-keys } ?f ((\text{fdom}' m_1 \triangleleft m_2) ++_f \{k \text{ \&\& } = v'\})$

using *dom-res-addition-in* by *fastforce*

also have  $\dots = \text{fmap-keys } ?f (\text{fdom}' m_1 \triangleleft m_2) ++_f \text{fmap-keys } ?f \{k \text{ \&\& } = v'\}$

proof –

from  $\langle m_1 \text{ \&\& } k = \text{None} \rangle$  have  $\text{fdom}' (\text{fdom}' m_1 \triangleleft m_2) \cap \text{fdom}' \{k \text{ \&\& } = v'\} = \{\}$

by (*simp add: fdom'-notI*)

then show *?thesis*

using *fmap-keys-hom* by *blast*

qed

also from *assms*

have  $\dots = \text{fmap-keys } ?f (\text{fdom}' m_1 \triangleleft m_2) ++_f \{k \text{ \&\& } = v' + v\}$

proof –

have  $\text{fmap-keys } ?f \{k \text{ \&\& } = v'\} = \{k \text{ \&\& } = v' + v\}$

proof (*intro fmap-ext*)

fix  $k'$

have  $\text{fmap-keys } ?f \{k \text{ \&\& } = v'\} \text{ \&\& } k' = \text{map-option } (?f k') (\{k \text{ \&\& } = v'\} \text{ \&\& } k')$

using *fmlookup-fmap-keys* .

also have  $\dots = \text{map-option } (?f k') (\text{if } k = k' \text{ then } \text{Some } v' \text{ else } \{\}\} \text{ \&\& } k')$

by *simp*

also have  $\dots = \{k \text{ \&\& } = v' + v\} \text{ \&\& } k'$

by (*cases k' = k*) *simp-all*

finally show  $\text{fmap-keys } ?f \{k \text{ \&\& } = v'\} \text{ \&\& } k' = \{k \text{ \&\& } = v' + v\} \text{ \&\& } k'$  .

qed

then show *?thesis*

by *simp*

qed

also have  $\dots = \text{fmap-keys } (\lambda k' v'. v' + m_1 \text{ \&\& } ! k') (\text{fdom}' m_1 \triangleleft m_2) ++_f \{k \text{ \&\& } = v' + v\}$

by (*simp add: fmap-ext*)

finally show *?thesis* .

qed

lemma *inter-plus-addition-notin*:

assumes  $m_1 \text{ \&\& } k = \text{None}$

and  $m_2 \text{ \&\& } k = \text{None}$

shows  $m_1(k \text{ \&\& } = v) \cap_+ m_2 = (m_1 \cap_+ m_2)$

proof –

let  $?f = \lambda k' v'. v' + m_1(k \text{ \&\& } = v) \text{ \&\& } ! k'$

from  $\langle m_2 \text{ \&\& } k = \text{None} \rangle$

have  $m_1(k \text{ \&\& } = v) \cap_+ m_2 = \text{fmap-keys } ?f (\text{fdom}' m_1 \triangleleft m_2)$

using *dom-res-addition-not-in* by *metis*

**also have**  $\dots = \text{fmmmap-keys } (\lambda k' v'. v' + m_1 \text{ \&\&! } k') (f\text{mdom}' m_1 \triangleleft m_2)$   
**proof** (*intro fmap-ext*)  
**fix**  $k'$   
**have**  $\text{fmmmap-keys } ?f (f\text{mdom}' m_1 \triangleleft m_2) \text{ \&\& } k' = \text{map-option } (?f k') ((f\text{mdom}' m_1 \triangleleft m_2) \text{ \&\& } k')$   
**using** *fmlookup-fmmmap-keys* .  
**also from**  $\langle m_1 \text{ \&\& } k = \text{None} \rangle$  **have**  $\dots = \text{fmmmap-keys } (\lambda k' v'. v' + m_1 \text{ \&\&! } k') (f\text{mdom}' m_1 \triangleleft m_2)$   
 $\text{ \&\& } k'$   
**by** (*cases k' = k*) (*simp-all add: fmdom'-notI*)  
**finally show**  $\text{fmmmap-keys } ?f (f\text{mdom}' m_1 \triangleleft m_2) \text{ \&\& } k' =$   
 $\text{fmmmap-keys } (\lambda k' v'. v' + m_1 \text{ \&\&! } k') (f\text{mdom}' m_1 \triangleleft m_2) \text{ \&\& } k' .$   
**qed**  
**finally show** *?thesis* .  
**qed**

**lemma** *union-plus-addition-notin*:

**assumes**  $m_1 \text{ \&\& } k = \text{None}$   
**and**  $m_2 \text{ \&\& } k = \text{None}$   
**shows**  $m_1(k \text{ \&\&:= } v) \cup_+ m_2 = (m_1 \cup_+ m_2) ++_f \{k \text{ \&\&:= } v\}$   
**proof** –  
**from**  $\langle m_2 \text{ \&\& } k = \text{None} \rangle$  **have**  $(m_1(k \text{ \&\&:= } v) \cup_+ m_2 =$   
 $f\text{mdom}' m_2 \triangleleft / m_1 ++_f \{k \text{ \&\&:= } v\} ++_f f\text{mdom}' (m_1(k \text{ \&\&:= } v)) \triangleleft / m_2 ++_f (m_1(k \text{ \&\&:= } v) \cap_+$   
 $m_2)$   
**by** (*simp add: fmdom'-notI*)  
**also from** *assms* **have**  $\dots =$   
 $f\text{mdom}' m_2 \triangleleft / m_1 ++_f \{k \text{ \&\&:= } v\} ++_f f\text{mdom}' m_1 \triangleleft / m_2 ++_f (m_1(k \text{ \&\&:= } v) \cap_+ m_2)$   
**by** (*smt (verit, best) Int-iff Int-insert-left-if0 fmdom'-fmupd fmdom'-notI fmfilter-cong'*)  
**also from** *assms* **have**  $\dots = f\text{mdom}' m_2 \triangleleft / m_1 ++_f \{k \text{ \&\&:= } v\} ++_f f\text{mdom}' m_1 \triangleleft / m_2 ++_f$   
 $(m_1 \cap_+ m_2)$   
**using** *inter-plus-addition-notin by metis*  
**also from** *assms* **have**  $\dots = f\text{mdom}' m_2 \triangleleft / m_1 ++_f f\text{mdom}' m_1 \triangleleft / m_2 ++_f (m_1 \cap_+ m_2) ++_f$   
 $\{k \text{ \&\&:= } v\}$   
**using** *fmap-singleton-comm*  
**by** (*metis (mono-tags, lifting) fmadd-assoc fmdom'-notI fmfilter-fmmmap-keys fmlookup-filter*)  
**finally show** *?thesis* .  
**qed**  
**end**