Appendix to "An algebraic approach to a Kripkean theory of probabilty and truth"—Chapter 2 of Gutpa and Belnaps 'The Revision Theory of Truth' and generalisations

Fabian Heimann

July 25, 2019

Contents

```
theory Chapter2-prob
  imports Main HOL.Zorn HOL.Order-Relation HOL.Real
  HOL−Library.Lattice-Syntax HOL−Library.Finite-Lattice
  HOL.Finite-Set
begin
```

1 Preliminary Matters

```
datatype bool2 = t2 | f2

fun leq2 :: ‹bool2 ⇒ bool2 ⇒ bool› where
‹leq2 t2 t2 = True› |
‹leq2 f2 f2 = True› |
‹leq2 - - = False›

lemma leq2-refl: leq2 b b = True by(cases b) auto
lemma leq2-antisym: ⟦ leq2 u v ; leq2 v u ⟧ ⟹ u = v by(cases u; cases v) auto
lemma leq2-trans: ⟦ leq2 u v; leq2 v w ⟧ ⟹ leq2 u w by(cases u; cases v; cases
w) auto

instantiation bool2 :: order
begin

definition less-eq-bool2-def: ‹ b1 ≤ b2 = leq2 b1 b2 ›
definition less-bool2-def: ‹ b1 < b2 = ((leq2 b1 b2) ∧ (b1 ≠ b2))›

instance proof
  fix x y :: bool2
  show (x < y) = (x ≤ y ∧ ¬ y ≤ x) unfolding less-eq-bool2-def less-bool2-def
by(cases x; cases y) auto
next
  fix x :: bool2
  from leq2-refl show  x ≤ x unfolding less-eq-bool2-def by auto
next
  fix x y z :: bool2
  from leq2-trans show  x ≤ y ⟹ y ≤ z ⟹ x ≤ z unfolding less-eq-bool2-def
by auto
next
  fix x y :: bool2
  from leq2-antisym show x ≤ y ⟹ y ≤ x ⟹ x = y unfolding less-eq-bool2-def
by auto
qed
end

datatype bool3 = n3 | t3 | f3

fun leq3 :: ‹bool3 ⇒ bool3 ⇒ bool› where
‹leq3 n3 n3 = True› |
‹leq3 t3 t3 = True› |
```

$\langle$*leq3 f3 f3* $=$ *True*$\rangle$ |
$\langle$*leq3 n3 t3* $=$ *True*$\rangle$ |
$\langle$*leq3 n3 f3* $=$ *True*$\rangle$ |
$\langle$*leq3 t3 n3* $=$ *False*$\rangle$ |
$\langle$*leq3 t3 f3* $=$ *False*$\rangle$ |
$\langle$*leq3 f3 n3* $=$ *False*$\rangle$ |
$\langle$*leq3 f3 t3* $=$ *False*$\rangle$

lemma *leq3-refl*: *leq3* (*b* :: *bool3*) *b* $=$ *True* **by**(*cases b*) *auto*
lemma *leq3-antisym*: $[\![$ *leq3 u v* ; *leq3 v u* $]\!] \Longrightarrow u = v$ **by**(*cases u*; *cases v*) *auto*
lemma *leq3-trans*: $[\![$ *leq3 u v*; *leq3 v w* $]\!] \Longrightarrow$ *leq3 u w* **by**(*cases u*; *cases v*; *cases w*) *auto*

instantiation *bool3* :: *order*
begin

definition *less-eq-bool3-def*: $\langle$ $b1 \leq b2 =$ *leq3 b1* (*b2* :: *bool3*) $\rangle$
definition *less-bool3-def*: $\langle$ $b1 < b2 =$ ((*leq3 b1* (*b2* :: *bool3*)) $\wedge$ ($b1 \neq b2$))$\rangle$

instance proof
  fix *x y* :: *bool3*
  show $(x < y) = (x \leq y \wedge \neg\ y \leq x)$ **unfolding** *less-eq-bool3-def less-bool3-def* **by**(*cases x*; *cases y*) *auto*
next
  fix *x* :: *bool3*
  from *leq3-refl* **show** $x \leq x$ **unfolding** *less-eq-bool3-def* **by** *auto*
next
  fix *x y z* :: *bool3*
  from *leq3-trans* **show** $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$ **unfolding** *less-eq-bool3-def* **by** *auto*
next
  fix *x y* :: *bool3*
  from *leq3-antisym* **show** $x \leq y \Longrightarrow y \leq x \Longrightarrow x = y$ **unfolding** *less-eq-bool3-def* **by** *auto*
qed
end

datatype *bool4* $=$ *n4* | *t4* | *f4* | *b4*

fun *leq4* :: $\langle$*bool4* $\Rightarrow$ *bool4* $\Rightarrow$ *bool*$\rangle$ **where**
$\langle$*leq4* - *b4* $=$ *True*$\rangle$ |
$\langle$*leq4 n4* - $=$ *True*$\rangle$ |
$\langle$*leq4 t4 n4* $=$ *False*$\rangle$ |
$\langle$*leq4 t4 t4* $=$ *True* $\rangle$ |
$\langle$*leq4 t4 f4* $=$ *False*$\rangle$ |
$\langle$*leq4 f4 n4* $=$ *False*$\rangle$ |
$\langle$*leq4 f4 t4* $=$ *False*$\rangle$ |
$\langle$*leq4 f4 f4* $=$ *True*$\rangle$ |
$\langle$*leq4 b4 n4* $=$ *False*$\rangle$ |

⟨*leq4 b4 t4* = *False*⟩ |
⟨*leq4 b4 f4* = *False*⟩

lemma *leq4-refl*: *leq4 b b* = *True* **by**(*cases b*) *auto*
lemma *leq4-antisym*: ⟦ *leq4 u v* ; *leq4 v u* ⟧ ⟹ *u* = *v* **by**(*cases u*; *cases v*) *auto*
lemma *leq4-trans*: ⟦ *leq4 u v*; *leq4 v w* ⟧ ⟹ *leq4 u w* **by**(*cases u*; *cases v*; *cases w*) *auto*

instantiation *bool4* :: *order*
begin

definition *less-eq-bool4-def*: ⟨ *b1* ≤ *b2* = *leq4 b1* (*b2* :: *bool4*) ⟩
definition *less-bool4-def*: ⟨ *b1* < *b2* = ((*leq4 b1* (*b2* :: *bool4*)) ∧ (*b1* ≠ *b2*))⟩

instance proof
  fix *x y* :: *bool4*
  show (*x* < *y*) = (*x* ≤ *y* ∧ ¬ *y* ≤ *x*) **unfolding** *less-eq-bool4-def less-bool4-def* **by**(*cases x*; *cases y*) *auto*
next
  fix *x* :: *bool4*
  from *leq4-refl* **show** *x* ≤ *x* **unfolding** *less-eq-bool4-def* **by** *auto*
next
  fix *x y z* :: *bool4*
  from *leq4-trans* **show** *x* ≤ *y* ⟹ *y* ≤ *z* ⟹ *x* ≤ *z* **unfolding** *less-eq-bool4-def* **by** *auto*
next
  fix *x y* :: *bool4*
  from *leq4-antisym* **show** *x* ≤ *y* ⟹ *y* ≤ *x* ⟹ *x* = *y* **unfolding** *less-eq-bool4-def* **by** *auto*
qed
end

fun *inf4* :: ⟨*bool4* ⇒ *bool4* ⇒ *bool4*⟩ **where**
inf4 n4 (*b* :: *bool4*) = *n4* |
inf4 (*b* :: *bool4*) *b4* = *b* |
inf4 (*b* :: *bool4*) *n4* = *n4* |
inf4 b4 (*b* :: *bool4*) = *b* |
inf4 t4 t4 = *t4* | *inf4 f4 f4* = *f4* |
inf4 t4 f4 = *n4* | *inf4 f4 t4* = *n4*

fun *sup4* :: ⟨*bool4* ⇒ *bool4* ⇒ *bool4*⟩ **where**
sup4 n4 (*b* :: *bool4*) = *b* |
sup4 (*b* :: *bool4*) *b4* = *b4* |
sup4 (*b* :: *bool4*) *n4* = *b* |
sup4 b4 (*b* :: *bool4*) = *b4* |
sup4 t4 t4 = *t4* | *sup4 f4 f4* = *f4* |
sup4 t4 f4 = *b4* | *sup4 f4 t4* = *b4*

instantiation *bool4* :: *semilattice-inf*

begin

definition *inf-bool4-def*: *inf b1* (*b2* :: *bool4*) = *inf4 b1 b2*

instance proof
  fix *x y* :: *bool4*
  show *inf x y* ≤ *x* **unfolding** *inf-bool4-def less-eq-bool4-def* **by**(*cases x*; *cases y*) *auto*
next
fix *x y* :: *bool4*
  show *inf x y* ≤ *y* **unfolding** *inf-bool4-def less-eq-bool4-def* **by**(*cases x*; *cases y*) *auto*
next
  fix *x y z* :: *bool4*
  show *x* ≤ *y* ⟹ *x* ≤ *z* ⟹ *x* ≤ *inf y z* **unfolding** *inf-bool4-def less-eq-bool4-def* **by**(*cases x*; *cases y*; *cases z*) *auto*
qed
end

instantiation *bool4* :: *semilattice-sup*
begin

definition *sup-bool4-def*: *sup b1* (*b2* :: *bool4*) = *sup4 b1 b2*

instance proof
  fix *x y* :: *bool4*
  show *x* ≤ *sup x y* **unfolding** *sup-bool4-def less-eq-bool4-def* **by**(*cases x*; *cases y*) *auto*
next
fix *x y* :: *bool4*
  show *y* ≤ *sup x y* **unfolding** *sup-bool4-def less-eq-bool4-def* **by**(*cases x*; *cases y*) *auto*
next
  fix *y x z* :: *bool4*
  show *y* ≤ *x* ⟹ *z* ≤ *x* ⟹ *sup y z* ≤ *x* **unfolding** *sup-bool4-def less-eq-bool4-def* **by**(*cases x*; *cases y*; *cases z*) *auto*
qed
end

instantiation *bool4* :: *lattice*
begin
instance proof qed
end

lemma *bool4-induct*: *P n4* ⟹ *P t4* ⟹ *P f4* ⟹ *P b4* ⟹ *P x* **by**(*cases x*) *auto*

lemma *UNIV-bool4*: *UNIV* = {*n4*, *t4*, *f4*, *b4*}
  proof(*auto intro*: *bool4-induct*)
    show ⋀*x*. *x* ≠ *n4* ⟹ *x* ≠ *t4* ⟹ *x* ≠ *f4* ⟹ *x* = *b4* **proof** −

fix x::$bool4$
show $x \neq n4 \Longrightarrow x \neq t4 \Longrightarrow x \neq f4 \Longrightarrow x = b4$ **by**(*cases x*) *auto*
qed
qed

instantiation *bool4* :: *finite*
begin
instance by *standard*(*simp add*: *UNIV-bool4*)
end

instantiation *bool4* :: *finite-lattice-complete*
begin

definition *Inf-bool4-def*: *Inf* (*X* :: *bool4 set*) = *Finite-Set.fold inf b4 X*
definition *Sup-bool4-def*: *Sup* (*X* :: *bool4 set*) = *Finite-Set.fold sup n4 X*

definition *bot-bool4-def*: *bot* = *n4*
definition *top-bool4-def*: *top* = *b4*

instance proof
show (*bot* :: *bool4*) = *Inf-fin UNIV*
by (*simp add*: *UNIV-bool4 bot-bool4-def inf-bool4-def*)
next
show (*top* :: *bool4*) = *Sup-fin UNIV*
by (*simp add*: *UNIV-bool4 sup-bool4-def top-bool4-def*)
next
fix *A* :: *bool4 set*
show $\bigsqcap A$ = *Finite-Set.fold* $(\sqcap)\ \top\ A$
by (*simp add*: *Inf-bool4-def top-bool4-def*)
next
fix *A* :: *bool4 set*
show $\bigsqcup A$ = *Finite-Set.fold* $(\sqcup)\ \bot\ A$
by (*simp add*: *Sup-bool4-def bot-bool4-def*)
qed
end

lemma ‹ $\bigsqcup$ {*n4*,*t4*,*f4*} = *b4*›
by (*metis Sup-UNIV Sup-insert UNIV-bool4 ccpo-Sup-singleton sup4.simps*(*12*) *sup4.simps*(*2*) *sup4.simps*(*3*) *sup-bool4-def top-bool4-def*)

abbreviation *upper-bound* :: $'a$::*order* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool* (**infix** $\gtrsim$ *60*) **where**
upper-bound x Y $\equiv \forall\ y \in Y.\ x \geq y$

abbreviation *lower-bound* :: $'a$::*order* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool* (**infix** $\lesssim$ *60*)**where**
lower-bound x Y $\equiv \forall\ y \in Y.\ x \leq y$

definition *supR* :: $'a$ ::*order* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool* **where**
supR $(x::'a)$ $(Y::'a\ set) \equiv (\ x \gtrsim Y\ ) \wedge (\forall\ y.\ y \gtrsim Y \longrightarrow x \leq y)$

definition *supRs* :: $'a$ ::*order* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool* **where**
supRs $(x{::}'a)$ $(Y{::}'a\ set)$ $(X :: 'a\ set)$ $\equiv (x \in X) \wedge (\ x \gtrsim Y\ ) \wedge (\forall\ y \in X.\ y \gtrsim Y \longrightarrow x \leq y)$

definition *infR* :: $'a$ ::*order* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool* **where**
infR $(x{::}'a)$ $(Y :: 'a\ set) \equiv (\ x \lesssim Y) \wedge (\forall\ y.\ y \lesssim Y \longrightarrow x \geq y)$

definition *infRs* :: $'a$ ::*order* $\Rightarrow$ $'a$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool* **where**
infRs $(x{::}'a)$ $(Y :: 'a\ set)$ $(X :: 'a\ set) \equiv (x \in X) \wedge (\ x \lesssim Y) \wedge (\forall\ y \in X.\ y \lesssim Y \longrightarrow x \geq y)$

definition *lattice* :: $('a$ ::*order*$)$ *set* $\Rightarrow$ *bool* **where**
lattice $X \equiv (\forall\ X' \subseteq X.\ X' \neq \{\} \wedge$ *finite* $X' \longrightarrow (\exists\ s.$ *supR* $s\ X') \wedge (\exists\ i.$ *infR* $i\ X')\ )$

definition *complete-latticeR* :: $('a$ ::*order*$)$ *set* $\Rightarrow$ *bool* **where**
complete-latticeR $X \equiv (\forall\ X' \subseteq X.\ (\exists\ s \in X.$ *supRs* $s\ X'\ X) \wedge (\exists\ i \in X.$ *infRs* $i\ X'\ X)\ )$

lemma *supRs-prop*: *supRs* $x\ Y\ X \Longrightarrow x \in X$ **by**(*simp add*: *supRs-def*)

lemma *complete-lattice-iff-sup*:
complete-latticeR $X \longleftrightarrow (\forall\ X' \subseteq X.\ (\exists\ s \in X.$ *supRs* $s\ X'\ X))$
proof
  assume *complete-latticeR* X
  from *this* **show** $\forall X'{\subseteq}X.\ \exists s{\in}X.$ *supRs* $s\ X'\ X$ **by**(*simp add*: *complete-latticeR-def*)
next
  assume *Hsup*: $\forall X'{\subseteq}X.\ \exists s{\in}X.$ *supRs* $s\ X'\ X$
  have $(\forall\ X' \subseteq X.\ (\exists\ s \in X.$ *supRs* $s\ X'\ X) \wedge (\exists\ i \in X.$ *infRs* $i\ X'\ X)\ )$ **proof**
    fix X'
    show $X' \subseteq X \longrightarrow (\exists\ s \in X.$ *supRs* $s\ X'\ X) \wedge (\exists\ i \in X.$ *infRs* $i\ X'\ X)$ **proof**
      assume *HX'*:$X' \subseteq X$
      show $(\exists\ s \in X.$ *supRs* $s\ X'\ X) \wedge (\exists\ i \in X.$ *infRs* $i\ X'\ X)$ **proof**
        from *HX' Hsup* **show** $(\exists\ s \in X.$ *supRs* $s\ X'\ X)$ **by** *auto*
      next
        have $\{y \in X.\ y \lesssim X'\} \subseteq X$ **by** *auto*
        from *this Hsup* **have** $\exists\ s \in X.$ *supRs* $s\ \{y \in X.\ y \lesssim X'\}\ X$ **by** *simp*
        then obtain s **where** *s1*: $s \in X$ **and** *s2*: *supRs* $s\ \{y \in X.\ y \lesssim X'\}\ X$ **by** *auto*
        from *s2* *supRs-def*[*of* $s\ \{y \in X.\ y \lesssim X'\}\ X$] **have** *s3*: $s \lesssim X'$
          using *HX'* **by** *blast*
        from *s2* *supRs-def*[*of* $s\ \{y \in X.\ y \lesssim X'\}\ X$] **have** *s4*: $\forall\ y \in X.\ y \lesssim X' \longrightarrow y \leq s$
          by *simp*
        show $\exists i{\in}X.$ *infRs* $i\ X'\ X$ **proof**
          from *s1 s3 s4* **show** *infRs* $s\ X'\ X$ **by**(*simp add*: *infRs-def*)
        next
          from *s1* **show** $s \in X$ **by** *auto*
        qed

    qed
  qed
qed
from *this* **show** *complete-latticeR X* **by** (*simp add*: *complete-latticeR-def*)
qed

lemma *complete-lattice-iff-inf*:
complete-latticeR X $\longleftrightarrow$ ($\forall$ $X' \subseteq X$. ($\exists$ $s \in X$. *infRs s X' X*))
proof
  assume *complete-latticeR X*
  from *this* **show** $\forall X' \subseteq X$. $\exists s \in X$. *infRs s X' X* **by**(*simp add*: *complete-latticeR-def*)
next
  assume *Hinf*: $\forall X' \subseteq X$. $\exists s \in X$. *infRs s X' X*
  have ($\forall$ $X' \subseteq X$. ($\exists$ $s \in X$. *supRs s X' X*) $\wedge$ ($\exists$ $i \in X$. *infRs i X' X*)) **proof**
    fix X'
    show $X' \subseteq X \longrightarrow$ ($\exists$ $s \in X$. *supRs s X' X*) $\wedge$ ($\exists$ $i \in X$. *infRs i X' X*) **proof**
      assume *HX'*:$X' \subseteq X$
      show ($\exists$ $s \in X$. *supRs s X' X*) $\wedge$ ($\exists$ $i \in X$. *infRs i X' X*) **proof**
        from *HX' Hinf* **show** ($\exists$ $s \in X$. *infRs s X' X*) **by** *auto*
      next
        have $\{y \in X.\ y \gtrsim X'\} \subseteq X$ **by** *auto*
        from *this Hinf* **have** $\exists$ $s \in X$. *infRs s* $\{y \in X.\ y \gtrsim X'\}$ *X* **by** *simp*
        then obtain *s* **where** *s1*: $s \in X$ **and** *s2*: *infRs s* $\{y \in X.\ y \gtrsim X'\}$ *X* **by** *auto*
        from *s2 infRs-def*[*of s* $\{y \in X.\ y \gtrsim X'\}$ *X*] **have** *s3*: $s \gtrsim X'$
          using *HX'* **by** *blast*
        from *s2 infRs-def*[*of s* $\{y \in X.\ y \gtrsim X'\}$ *X*] **have** *s4*: $\forall$ $y \in X.\ y \gtrsim X' \longrightarrow y \geq s$
          by *simp*
        show $\exists i \in X$. *supRs i X' X* **proof**
          from *s1 s3 s4* **show** *supRs s X' X* **by**(*simp add*: *supRs-def*)
        next
          from *s1* **show** $s \in X$ **by** *auto*
        qed
      qed
    qed
  qed
  from *this* **show** *complete-latticeR X* **by** (*simp add*: *complete-latticeR-def*)
qed

lemma *order-example1*: *supRs t3* {*t3*, *n3*} {*t3*, *n3*, *f3*}
  by(*simp add*: *supRs-def less-eq-bool3-def*)

lemma *order-example1B*: *supRs f3* {*f3*, *n3*} {*t3*, *n3*, *f3*}
  by(*simp add*: *supRs-def less-eq-bool3-def*)

lemma *order-example2*: *infR n3* {*t3*, *n3*}
  by (*simp add*: *infR-def less-eq-bool3-def*)

lemma *order-example3*: *infR n3* $\{t3, f3\}$
by (*smt bool3.exhaust infR-def insertI1 insert-subset leq3.simps*(*1*) *leq3.simps*(*4*) *leq3.simps*(*5*) *leq3.simps*(*7*) *leq3.simps*(*9*) *less-eq-bool3-def subset-insertI*)

lemma *order-example4*: $\neg$ *supR b* $\{t3, f3\}$
by(*cases b*; *simp add*: *supR-def less-eq-bool3-def*)

lemma *order-example5*: $\neg$ *upper-bound b* $\{t3, f3\}$
by(*cases b*; *simp add*: *less-eq-bool3-def*)

context *order*
begin
definition *consi* :: ‹′*a set* ⇒ ′*a set* ⇒ *bool*› **where**
‹*consi Y X* $\longleftrightarrow$ ($\forall$ $x \in Y$. $\forall$ $y \in Y$. $\exists$ $b \in X$. ($x \leq b \wedge y \leq b$))›
end

lemma *consi-subset*: $[\![X' \subseteq X;$ *consi Y X*′$]\!] \Longrightarrow$ *consi Y X*
by (*smt consi-def subset-eq*)

class *Vccpo* = *order* + *Sup* +
assumes *Vccpo-Sup-upper* : ‹*consi A UNIV* $\Longrightarrow x \in A \Longrightarrow x \leq$ *Sup A*›
assumes *Vccpo-Sup-least*: ‹*consi A UNIV* $\Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow x \leq z) \Longrightarrow$ *Sup A* $\leq z$›

fun *sup3* :: ‹*bool3* ⇒ *bool3* ⇒ *bool3*› **where**
sup3 n3 (*b* :: *bool3*) = *b* |
sup3 (*b* :: *bool3*) *n3* = *b* |
sup3 t3 t3 = *t3* | *sup3 f3 f3* = *f3* |
sup3 t3 f3 = *undefined* | *sup3 f3 t3* = *undefined*

lemma *bool3-induct*: *P n3* $\Longrightarrow$ *P t3* $\Longrightarrow$ *P f3* $\Longrightarrow$ *P x* **by**(*cases x*) *auto*

lemma *UNIV-bool3*: *UNIV* = $\{n3, t3, f3\}$
proof(*auto intro*: *bool3-induct*)
show $\bigwedge x.\ x \neq n3 \Longrightarrow x \neq t3 \Longrightarrow x = f3$ **proof** −
fix *x*::*bool3*
show $x \neq n3 \Longrightarrow x \neq t3 \Longrightarrow x = f3$ **by**(*cases x*) *auto*
qed
qed

instantiation *bool3* :: *finite*
begin
instance by *standard*(*simp add*: *UNIV-bool3*)
end

lemma *bool3-set-cases*:
((*X*::*bool3 set*) = {} $\vee$ *X* = $\{t3\}$ $\vee$ *X* = $\{f3\}$ $\vee$ *X* = $\{n3\}$ $\vee$ *X* = $\{t3, f3\}$ $\vee$ *X*

$= \{t3,\ n3\} \vee X = \{f3,\ n3\} \vee X = \{t3,\ f3,\ n3\})$
proof −
fix X :: ‹*bool3 set*›
have $I1$: *finite* X **by** *auto*
have $I2$: $\{\} = \{\} \vee \{\} = \{t3\} \vee \{\} = \{f3\} \vee \{\} = \{n3\} \vee \{\} = \{t3,\ f3\} \vee \{\} = \{t3,\ n3\} \vee \{\} = \{f3,\ n3\} \vee \{\} = \{t3,\ f3,\ n3\}$ **by** *auto*
have $I3$: $(\bigwedge x\ F.\ finite\ F \Longrightarrow x \notin F \Longrightarrow F = \{\} \vee F = \{t3\} \vee F = \{f3\} \vee F = \{n3\} \vee F = \{t3,\ f3\} \vee F = \{t3,\ n3\} \vee F = \{f3,\ n3\} \vee F = \{t3,\ f3,\ n3\} \Longrightarrow (insert\ x\ F = \{\} \vee insert\ x\ F = \{t3\} \vee insert\ x\ F = \{f3\} \vee insert\ x\ F = \{n3\} \vee insert\ x\ F = \{t3,\ f3\} \vee insert\ x\ F = \{t3,\ n3\} \vee insert\ x\ F = \{f3,\ n3\} \vee insert\ x\ F = \{t3,\ f3,\ n3\}))$ **proof** −
fix x :: *bool3*
fix F :: *bool3 set*
assume $A1$: *finite* F
assume $A2$: $x \notin F$
assume $A3$: $F = \{\} \vee F = \{t3\} \vee F = \{f3\} \vee F = \{n3\} \vee F = \{t3,\ f3\} \vee F = \{t3,\ n3\} \vee F = \{f3,\ n3\} \vee F = \{t3,\ f3,\ n3\}$
show $(insert\ x\ F = \{\} \vee insert\ x\ F = \{t3\} \vee insert\ x\ F = \{f3\} \vee insert\ x\ F = \{n3\} \vee insert\ x\ F = \{t3,\ f3\} \vee insert\ x\ F = \{t3,\ n3\} \vee insert\ x\ F = \{f3,\ n3\} \vee insert\ x\ F = \{t3,\ f3,\ n3\})$
by (*smt A2 A3 bool3.exhaust insertI1 insert-commute*)
qed
from *I1 I2 I3 finite-induct*[*of* X $(\lambda\ x.\ x=\{\} \vee x = \{t3\} \vee x = \{f3\} \vee x = \{n3\} \vee x = \{t3,\ f3\} \vee x = \{t3,\ n3\} \vee x = \{f3,\ n3\} \vee x = \{t3,\ f3,\ n3\})$]
show $X = \{\} \vee X = \{t3\} \vee X = \{f3\} \vee X = \{n3\} \vee X = \{t3,\ f3\} \vee X = \{t3,\ n3\} \vee X = \{f3,\ n3\} \vee X = \{t3,\ f3,\ n3\}$ **by** *auto*
qed

instantiation *bool3* :: *Sup*
begin
definition *Sup-bool3-def*:
Sup (*X* :: *bool3 set*) = *Finite-Set.fold sup3 n3 X*
instance proof qed
end

2 Coherent Complete Partial Orders

definition *ccpo* ::‹(*'a* ::*order*) *set* ⇒ *bool*› **where**
$ccpo\ X \equiv (\ \forall\ X' \subseteq X.\ (consi\ X'\ X) \longrightarrow (\exists\ \ b \in X.\ supRs\ b\ X'\ X)\ )$

lemma *empty-consi*: *consi* {} *X* **by**(*simp add*: *consi-def*)

lemma *C1*: *consi* { *b* :: *bool3*} {*n3*, *t3*, *f3*}
using *bool3.exhaust consi-def* **by** *blast*

lemma *C2*: *consi* $\{n3, t3\}$ $\{n3, t3, f3\}$
  by (*metis consi-def doubleton-eq-iff insert-subset order-example1 subset-insertI supRs-def*)

lemma *C3*: *consi* $\{n3, f3\}$ $\{n3, t3, f3\}$
  by (*metis consi-def doubleton-eq-iff insert-subset order-example1B subset-insertI supRs-def*)

lemma *C4-helper*: $\neg$ ($\exists$ *b*::*bool3*. $b \gtrsim \{t3, f3\}$)
  using *order-example5* **by** *blast*

lemma *C4*: $\neg$ *consi* $\{t3, f3\}$ $\{n3, t3, f3\}$ **proof**
  assume *consi* $\{t3, f3\}$ $\{n3, t3, f3\}$
  from *this* **have** $\exists\ b \in \{n3, t3, f3\}.\ b \geq t3 \wedge b \geq f3$
    by (*simp add*: *consi-def*)
  from *this C4-helper* **show** *False* **by** *auto*
  qed

lemma *C5*: $\neg$ *consi* $\{t3, f3, n3\}$ $\{n3, t3, f3\}$
  using *C4* **by** (*simp add*: *consi-def*; *blast*)

lemma $\neg$ *ccpo* $\{\}$ **by**(*simp add*: *ccpo-def consi-def*)

lemma *complete-lattice-implies-ccpo*:
complete-latticeR (X :: ('*a* :: *order*) *set*) $\Longrightarrow$ *ccpo* X
  by (*simp add*: *ccpo-def complete-lattice-iff-sup*)

lemma *ccpo-least-element*:
ccpo (X :: ('*a* ::*order*) *set*) $\Longrightarrow \exists\, l{\in}X.\ l \lesssim X$
  by (*metis ccpo-def empty-iff empty-subsetI supRs-def consi-def*)

lemma *ccpo-nonempty-has-inf*:
$[\![$ *ccpo* (X :: ('*a* ::*order*) *set*) ; $X' \subseteq X$; $X' \neq \{\}$ $]\!]$
    $\Longrightarrow$ ($\exists\ i \in X.$ *infRs* i X' X)
proof $-$
  fix X :: '*a set*
  assume *H1*: *ccpo* X
  fix Y
  assume *H2*: $Y \subseteq X$
  assume *H3*: $Y \neq \{\}$

  from *H3 H2* **have** *C*: *consi* $\{u \in X.\ u \lesssim Y\}$ X **by**(*simp add*: *consi-def*; *blast*)
  have $\{u \in X.\ u \lesssim Y\} \subseteq X$ **by**(*auto*)
  from *H1 C this ccpo-def* **have** $\exists\ b \in X.$ *supRs* b $\{u \in X.\ u \lesssim Y\}$ X **by**(*blast*)
  then obtain b **where** *HbinX*: $b \in X$ **and** *Hbsup*: *supRs* b $\{u \in X.\ u \lesssim Y\}$ X **by** *auto*

  have *infH1*: $b \lesssim Y$ **proof**
    fix y

```
    assume Hy: y ∈ Y
    from this have y ≳ {u ∈ X. u ≲ Y} by auto
    from this Hbsup HbinX show b ≤ y
      by (meson H2 Hy subsetCE supRs-def)
  qed

  have infH2: ∀ b2 ∈ X. b2 ≲ Y ⟶ b2 ≤ b proof
    fix b2
    assume Hb2: b2 ∈ X
    show b2 ≲ Y ⟶ b2 ≤ b proof
      assume H: b2 ≲ Y
      from this Hb2 have b2 ∈ {u ∈ X. u ≲ Y} by auto
      from this Hbsup show b2 ≤ b by (simp add: supRs-def)
    qed
  qed

  show (∃ i∈X. infRs i Y X) proof
    from HbinX show b ∈ X by auto
  next
    from HbinX infH1 infH2 show infRs b Y X by (simp add: infRs-def)
  qed
qed

lemma greatest-element-implies-complete-lattice:
⟦ ccpo (X :: ('a::order) set); ∃ g ∈ X. g ≳ X⟧ ⟹ complete-latticeR X
proof −
  fix X :: 'a set
  assume H1: ccpo X
  assume H2: ∃ g ∈ X. g ≳ X
  from H2 obtain g where Hg1: g ∈ X and Hg2: g ≳ X by auto

  have (∀ X' ⊆ X. (∃ s ∈ X. infRs s X' X)) proof
    fix X'
    show X' ⊆ X ⟶ (∃ s∈X. infRs s X' X) proof
      assume H3: X' ⊆ X
      have A1: (X' = {}) ⟶ (∃ s∈X. infRs s X' X) proof
        assume H4: X' = {}
        show ∃ s∈X. infRs s X' X proof
          show infRs g X' X
            by (simp add: Hg1 H4 Hg2 infRs-def)
        next
          from Hg1 show g ∈ X by auto
        qed
      qed
      have A2: X' ≠ {} ⟹ ∃ s∈X. infRs s X' X
        by (simp add: ccpo-nonempty-has-inf H1 H3)

      from A1 A2 show (∃ s∈X. infRs s X' X) by auto
    qed
```

qed
from *complete-lattice-iff-inf this* **show** *complete-latticeR X* **by** *auto*
qed

lemma *b3-ccpo*: *ccpo* $\{n3, t3, f3\}$
proof −
have $\forall X' \subseteq \{n3, t3, f3\}$.
$(consi\ X'\ \{n3, t3, f3\}) \longrightarrow (\exists b{\in}\{n3, t3, f3\}.\ supRs\ b\ X'\ \{n3, t3, f3\})$
proof
fix X' :: *bool3 set*
show $X' \subseteq \{n3, t3, f3\} \longrightarrow (consi\ X'\ \{n3, t3, f3\}) \longrightarrow$
$(\exists b{\in}\{n3, t3, f3\}.\ supRs\ b\ X'\ \{n3, t3, f3\})$ **proof**
assume *H*: $X' \subseteq \{n3, t3, f3\}$
from *bool3-set-cases* **have** *C*: $X' = \{\} \vee X' = \{t3\} \vee X' = \{f3\} \vee X' = \{n3\} \vee X' = \{t3, f3\} \vee X' = \{t3, n3\} \vee X' = \{f3, n3\} \vee X' = \{t3, f3, n3\}$ **by** *auto*
then show $(consi\ X'\ \{n3, t3, f3\}) \longrightarrow (\exists b{\in}\{n3, t3, f3\}.\ supRs\ b\ X'\ \{n3, t3, f3\})$ **proof**
assume $X' = \{\}$
from *this* **have** *supRs n3* X' $\{n3, t3, f3\}$
by (*simp add*: *less-eq-bool3-def supRs-def*)
from *this* **show** *?thesis* **by** *auto*
next
assume $X' = \{t3\} \vee X' = \{f3\} \vee X' = \{n3\} \vee X' = \{t3, f3\} \vee$
$X' = \{t3, n3\} \vee X' = \{f3, n3\} \vee X' = \{t3, f3, n3\}$
then show *?thesis* **proof**
assume $X' = \{t3\}$
from *this* **have** *supRs t3* X' $\{n3, t3, f3\}$ **by** (*simp add*: *supRs-def*)
from *this* **show** *?thesis* **by** *auto*
next
assume $X' = \{f3\} \vee X' = \{n3\} \vee X' = \{t3, f3\} \vee X' = \{t3, n3\} \vee$
$X' = \{f3, n3\} \vee X' = \{t3, f3, n3\}$
then show *?thesis* **proof**
assume $X' = \{f3\}$
from *this* **have** *supRs f3* X' $\{n3, t3, f3\}$ **by** (*simp add*: *supRs-def*)
from *this* **show** *?thesis* **by** *auto*
next
assume $X' = \{n3\} \vee X' = \{t3, f3\} \vee X' = \{t3, n3\} \vee X' = \{f3, n3\} \vee$
$X' = \{t3, f3, n3\}$
then show *?thesis* **proof**
assume $X' = \{n3\}$
from *this* **have** *supRs n3* X' $\{n3, t3, f3\}$ **by** (*simp add*: *supRs-def*)
from *this* **show** *?thesis* **by** *auto*
next
assume $X' = \{t3, f3\} \vee X' = \{t3, n3\} \vee X' = \{f3, n3\} \vee X' = \{t3, f3, n3\}$
then show *?thesis* **proof**
assume $X' = \{t3, f3\}$
from *this* **have** $\neg$ *consi* X' $\{n3,t3,f3\}$ **using** *C4 UNIV-bool3* **by** *auto*

from *this* **show** *?thesis* **using** *UNIV-bool3* **by** *blast*
next
assume $X' = \{t3, n3\} \vee X' = \{f3, n3\} \vee X' = \{t3, f3, n3\}$
then show *?thesis* **proof**
assume $X' = \{t3, n3\}$
from *this* **have** *supRs t3* X' $\{n3, t3, f3\}$ **using** *order-example1*
by (*simp add*: *insert-commute*)
from *this* **show** *?thesis* **by** *auto*
next
assume $X' = \{f3, n3\} \vee X' = \{t3, f3, n3\}$
then show *?thesis* **proof**
assume $X' = \{f3, n3\}$
from *this* **have** *supRs f3* X' $\{t3, n3, f3\}$ **using** *order-example1B*
by *blast*
from *this* **show** *?thesis*
by (*simp add*: *insert-commute*)
next
assume $X' = \{t3, f3, n3\}$
from *this* **have** ¬ *consi* X' $\{n3,t3,f3\}$ **using** *C4 UNIV-bool3* **by**
(*simp add*: *consi-def*; *auto*)
from *this* **show** *?thesis* **using** *UNIV-bool3* **by** *blast*
qed
qed
qed
qed
qed
qed
qed
qed
qed
from *this ccpo-def*[*of* $\{n3,t3,f3\}$] **show** *?thesis* **by** *auto*
qed

lemma *b4-complete-lattice*: *complete-latticeR* $\{b4, n4, t4, f4\}$
proof −
have $(\forall\ X' \subseteq \{b4, n4, t4, f4\}.\ (\exists\ s \in \{b4, n4, t4, f4\}.$ *supRs s* X' $\{b4, n4,$
$t4, f4\}))$ **proof**
fix X' :: *bool4 set*
show $X' \subseteq \{b4, n4, t4, f4\} \longrightarrow (\exists\, s \in \{b4, n4, t4, f4\}.$ *supRs s* X' $\{b4, n4,$
$t4, f4\})$ **proof**
assume *HX'*: $X' \subseteq \{b4, n4, t4, f4\}$
show $(\exists\, s \in \{b4, n4, t4, f4\}.$ *supRs s* X' $\{b4, n4, t4, f4\})$ **proof**
show *supRs* $(\bigsqcup X')$ X' $\{b4, n4, t4, f4\}$ **proof**−
have *Supp1*: $\bigsqcup X' \in \{b4, n4, t4, f4\}$ **using** *bool4.exhaust* **by** *blast*
have *Supp2*: $\bigsqcup X' \gtrsim X'$ **by** (*simp add*: *Sup-upper*)
have *Supp3*: $(\forall\ y \in \{b4, n4, t4, f4\}.\ y \gtrsim X' \longrightarrow \bigsqcup X' \leq y)$
by (*simp add*: *Sup-least*)
from *Supp1 Supp2 Supp3* **show** *?thesis* **by** (*simp add*: *supRs-def*)
qed

next
show $\bigsqcup X' \in \{b4, n4, t4, f4\}$ **using** *bool4.exhaust* **by** *blast*
qed
qed
qed
from *this* **show** *?thesis* **using** *complete-lattice-iff-sup* **by** *blast*
qed

lemma *complete-lattice-ccpo*:
fixes X ::($'a$ :: *complete-lattice*) *set*
assumes I:X = *UNIV*
shows *ccpo X*
proof −
have $\forall X' \subseteq X$. *consi* X' X $\longrightarrow$ ($\exists b \in X$. *supRs b* X' X) **proof**
fix X'
show $X' \subseteq X \longrightarrow$ *consi* X' X $\longrightarrow$ ($\exists b \in X$. *supRs b* X' X) **proof**
assume H: $X' \subseteq X$
show *consi* X' X $\longrightarrow$ ($\exists b \in X$. *supRs b* X' X) **proof**
assume *H2*: *consi* X' X
have $\exists b$. *supRs b* X' X **proof**
have (*Sup* X') $\in X$ **using** *assms* **by** *blast*
then show *supRs* (*Sup* X') X' X
by (*simp add*: *Sup-least Sup-upper supRs-def*)
qed
then show $\exists b \in X$. *supRs b* X' X **using** *assms* **by** *blast*
qed
qed
qed
then show *?thesis* **by** (*simp add*: *ccpo-def*)
qed

lemma *b4-ccpo*: *ccpo* $\{n4,t4,f4,b4\}$
by (*simp add*: *UNIV-bool4 complete-lattice-ccpo*)

lemma $\neg$ *ccpo* $\{t2,f2\}$
using *ccpo-least-element less-eq-bool2-def* **by** *fastforce*

lemma *above-sublattice-is-ccpo*:
$[\![$ *ccpo* (X :: ($'a$::*order*) *set*); $x \in X$ $]\!] \Longrightarrow$ *ccpo* $\{y \in X.\ x \le y\}$
proof −
fix X :: ($'a$ ::*order*) *set*
assume *H1*: *ccpo X*
fix x :: $'a$
assume *HxinX*: $x \in X$
have $\forall X' \subseteq \{y \in X.\ x \le y\}$. (*consi* X' $\{y \in X.\ x \le y\}$) $\longrightarrow$ ($\exists\ b \in \{y \in X.\ x \le y\}$. *supRs b* X' $\{y \in X.\ x \le y\}$) **proof**
fix X' ::$'a$ *set*
show $X' \subseteq \{y \in X.\ x \le y\} \longrightarrow$ (*consi* X' $\{y \in X.\ x \le y\}$) $\longrightarrow$ ($\exists\ b \in \{y \in X.\ x \le y\}$. *supRs b* X' $\{y \in X.\ x \le y\}$) **proof**

assume *H2*: $X' \subseteq \{y \in X.\ x \leq y\}$
show $(consi\ X'\ \{y \in X.\ x \leq y\}) \longrightarrow (\exists\ b \in \{y \in X.\ x \leq y\}.\ supRs\ b\ X'\ \{y \in X.\ x \leq y\})$ **proof**
assume *H3*: $(consi\ X'\ \{y \in X.\ x \leq y\})$
from *this* **have** *HX'1*:$(consi\ X'\ X)$ **by**(*simp add*: *consi-def*; *auto*)
from *H1 ccpo-def*[*of X*] **have** *H3*:$\forall X' \subseteq X.\ consi\ X'\ X \longrightarrow (\exists b \in X.\ supRs\ b\ X'\ X)$ **by** *auto*
from *H2* **have** *HX'2*: $X' \subseteq X$ **by** *auto*
from *HX'1 HX'2 H3* **have** $\exists\ b \in X.\ supRs\ b\ X'\ X$ **by** *auto*
then obtain b **where** *Hb1*: $b \in X$ **and** *Hb2*: $supRs\ b\ X'\ X$ **by** *auto*
show $\exists b \in \{y \in X.\ x \leq y\}.\ supRs\ b\ X'\ \{y \in X.\ x \leq y\}$ **proof**(*cases* $X' = \{\}$)
case *True*
show *?thesis* **proof**
have $x \leq x$ **by** *auto*
from *HxinX this* **show** $x \in \{y \in X.\ x \leq y\}$ **by** *simp*
then show $supRs\ x\ X'\ \{y \in X.\ x \leq y\}$ **by** (*simp add*: *True supRs-def*)
qed
next
case *False*
show *?thesis* **proof**
have $x \leq b$
by (*metis* (*no-types*, *lifting*) *Ball-Collect False H2 Hb2 bot.extremum-uniqueI order-trans subset-emptyI supRs-def*)
from *Hb2 this* **show** $supRs\ b\ X'\ \{y \in X.\ x \leq y\}$ **by** (*simp add*: *False supRs-def*)
next
show $b \in \{y \in X.\ x \leq y\}$
by (*metis* (*no-types*, *lifting*) *Collect-mem-eq Collect-mono-iff False H2 Hb1 Hb2 bot.extremum-uniqueI mem-Collect-eq order-trans subset-emptyI supRs-def*)
qed
qed
qed
qed
qed
then show *ccpo* $\{y \in X.\ x \leq y\}$ **by**(*simp add*: *ccpo-def*)
qed

lemma *below-sublattice-is-complete*:
$\llbracket\ ccpo\ (X :: ('a :: order)\ set);\ x \in X \rrbracket \Longrightarrow$
complete-latticeR $\{z \in X.\ z \leq x\}$
proof−
fix X :: *'a set*
assume *H1*: *ccpo X*
fix x :: *'a*
assume *HxinX*: $x \in X$
have $\forall X' \subseteq \{z \in X.\ z \leq x\}.(\exists\ i.\ i \in X \wedge i \leq x \wedge infRs\ i\ X'\ \{z \in X.\ z \leq x\})$
proof
fix X'

show $(X' \subseteq \{z \in X.\ z \leq x\}) \longrightarrow (\exists\, i.\ i \in X \wedge i \leq x \wedge \mathit{infRs}\ i\ X'\ \{z \in X.\ z \leq x\})$ **proof**
assume *H2*: $X' \subseteq \{z \in X.\ z \leq x\}$
show $(\exists\, i.\ i \in X \wedge i \leq x \wedge \mathit{infRs}\ i\ X'\ \{z \in X.\ z \leq x\})$ **proof**(*cases* $X' = \{\}$)
case *True*
show *?thesis* **proof**
from *True* **have** *H3*: $x \lesssim X' \wedge (\forall\, y \in \{z \in X.\ z \leq x\}.\ y \lesssim X' \longrightarrow y \leq x)$ **by** *simp*
show $x \in X \wedge x \leq x \wedge \mathit{infRs}\ x\ X'\ \{z \in X.\ z \leq x\}$ **proof**
from *HxinX* **show** $x \in X$ **by** *auto*
next
show $x \leq x \wedge \mathit{infRs}\ x\ X'\ \{z \in X.\ z \leq x\}$ **proof**
show $x \leq x$ **by** *auto*
next
from *HxinX H3* **show** $\mathit{infRs}\ x\ X'\ \{z \in X.\ z \leq x\}$ **by** (*simp add*: *infRs-def*)
qed
qed
qed
next
case *False*
from *this H1 H2 ccpo-nonempty-has-inf*[*of X X'*] **have**
$\exists\, i \in X.\ \mathit{infRs}\ i\ X'\ X$ **by** *auto*
from *this* **obtain** i **where** *Hi1*: $i \in X$ **and** *Hi2*: $\mathit{infRs}\ i\ X'\ X$ **by** *auto*
show *?thesis* **proof**
show $i \in X \wedge i \leq x \wedge \mathit{infRs}\ i\ X'\ \{z \in X.\ z \leq x\}$ **proof**
from *Hi1* **show** $i \in X$ **by** *auto*
next
show $i \leq x \wedge \mathit{infRs}\ i\ X'\ \{z \in X.\ z \leq x\}$ **proof**
from *Hi2 infRs-def* **show** $i \leq x$
by (*metis* (*no-types*, *lifting*) *Ball-Collect False H2 bot.extremum-uniqueI order-trans subset-emptyI*)
next
from *Hi2 infRs-def* **have** $i \leq x$
by (*metis* (*no-types*, *lifting*) *Ball-Collect False H2 bot.extremum-uniqueI order-trans subset-emptyI*)
from *Hi2 this* **show** $\mathit{infRs}\ i\ X'\ \{z \in X.\ z \leq x\}$ **by**(*simp add*: *infRs-def*)
qed
qed
qed
qed
qed
qed
from *this* **show** *complete-latticeR* $\{z \in X.\ z \leq x\}$ **by**(*simp add*: *complete-lattice-iff-inf complete-latticeR-def*)
qed

definition *SupFun* :: $('a :: \mathit{order})\ \mathit{set} \Rightarrow 'a\ \mathit{set} \Rightarrow 'a$ **where**

SupFun X Y = (*if* (*consi X Y*) *then* (*SOME x*. *supRs x X Y*) *else undefined*)

lemma *SupFun-inX*: $[\![$ *ccpo* (*Y* :: ('*a* :: *order*) *set*); (*X*:: '*a set*) $\subseteq$ *Y*;
consi X Y $]\!] \Longrightarrow$ (*SupFun X Y*) $\in$ *Y*
proof −
fix *X Y* :: ‹'*a set*›
assume *H1*: *ccpo Y*
assume *H2*: *X* $\subseteq$ *Y*
assume *H3*: *consi X Y*
show (*SupFun X Y*) $\in$ *Y* **proof**−
have (*SOME x*. ($x \in Y$) $\wedge$ ($\forall y{\in}X.\ y \le x$) $\wedge$ ($\forall y{\in}Y.\ (\forall ya{\in}X.\ ya \le y) \longrightarrow x \le y$)) $\in$ *Y* **proof**(*rule someI2-ex*)
from *H1 H2 H3* **show** $\exists a.\ a \in Y \wedge (\forall y{\in}X.\ y \le a) \wedge (\forall y{\in}Y.\ (\forall ya{\in}X.\ ya \le y) \longrightarrow a \le y)$
by(*simp add*: *ccpo-def supRs-def*) *blast*
next
show $\bigwedge x.\ x \in Y \wedge (\forall y{\in}X.\ y \le x) \wedge (\forall y{\in}Y.\ (\forall ya{\in}X.\ ya \le y) \longrightarrow x \le y) \Longrightarrow x \in Y$ **by** *auto*
qed
from *H3 this* **have** (*if consi X Y then* (*SOME x*. *supRs x X Y*) *else undefined*) $\in$ *Y* **by**(*simp add*: *supRs-def*)
then show *SupFun X Y* $\in$ *Y* **by**(*simp add*: *SupFun-def*)
qed
qed

lemma *SupFun-greater*: $[\![$ *ccpo* (*Y* :: ('*a* :: *order*) *set*); (*X*:: '*a set*) $\subseteq$ *Y*;
consi X Y $]\!] \Longrightarrow$ (*SupFun X Y*) $\gtrsim$ *X*
proof −
fix *X Y* :: ‹'*a set*›
assume *H1*: *ccpo Y*
assume *H2*: *X* $\subseteq$ *Y*
assume *H3*: *consi X Y*
show (*SupFun X Y*) $\gtrsim$ *X* **proof**
fix *y*
show $y \in X \Longrightarrow y \le$ *SupFun X Y* **proof**−
assume *H4*: $y \in X$
have $y \le$ (*SOME x*. ($x \in Y$) $\wedge$ ($\forall y{\in}X.\ y \le x$) $\wedge$ ($\forall y{\in}Y.\ (\forall ya{\in}X.\ ya \le y) \longrightarrow x \le y$)) **proof**(*rule someI2-ex*)
from *H1 H2 H3* **show** $\exists a.\ (a \in Y) \wedge (\forall y{\in}X.\ y \le a) \wedge (\forall y{\in}Y.\ (\forall ya{\in}X.\ ya \le y) \longrightarrow a \le y)$ **by**(*simp add*: *ccpo-def supRs-def*) *blast*
next
from *H4* **show** $\bigwedge x.\ (x \in Y) \wedge (\forall y{\in}X.\ y \le x) \wedge (\forall y{\in}Y.\ (\forall ya{\in}X.\ ya \le y) \longrightarrow x \le y) \Longrightarrow y \le x$ **by** *auto*
qed
from *H3 this* **have** $y \le$ (*if consi X Y then* (*SOME x*. *supRs x X Y*) *else undefined*) **by**(*simp add*: *supRs-def*)
then show $y \le$ *SupFun X Y* **by**(*simp add*: *SupFun-def*)
qed
qed

qed

lemma *SupFun-least*: $[\![$ *ccpo* $(Y :: ('a :: order)\ set)$; $(X :: 'a\ set) \subseteq Y$;
consi $X\ Y]\!] \Longrightarrow (\bigwedge y.\ y \in Y \Longrightarrow y \gtrsim X \Longrightarrow (SupFun\ X\ Y)$
$\leq y)$
proof $-$
fix $X\ Y :: \langle 'a\ set\rangle$ **fix** $y :: 'a$
assume *H1*: *ccpo Y*
assume *H2*: $X \subseteq Y$
assume *H3*: *consi X Y*
show $(\bigwedge y.\ y \in Y \Longrightarrow y \gtrsim X \Longrightarrow (SupFun\ X\ Y) \leq y)$ **proof**$-$
have $(\bigwedge y.\ y \in Y \Longrightarrow y \gtrsim X \Longrightarrow (SOME\ x.\ (x \in Y) \wedge (\forall y{\in}X.\ y \leq x) \wedge$
$(\forall y{\in}Y.\ y \gtrsim X \longrightarrow x \leq y)) \leq y)$ **proof**(*rule someI2-ex*)
from *H1 H2 H3* **show** $\bigwedge y.\ y \in Y \Longrightarrow \forall ya{\in}X.\ ya \leq y \Longrightarrow \exists a.\ (a \in Y)$
$\wedge (\forall y{\in}X.\ y \leq a) \wedge$
$(\forall y{\in}Y.(\forall ya{\in}X.\ ya \leq y) \longrightarrow a \leq y)$ **by**(*simp add*: *ccpo-def supRs-def*)
blast
next
show $\bigwedge y\ x.\ y \in Y \Longrightarrow \forall ya{\in}X.\ ya \leq y \Longrightarrow x \in Y \wedge (\forall y{\in}X.\ y \leq x) \wedge$
$(\forall y{\in}Y.\ (\forall ya{\in}X.\ ya \leq y) \longrightarrow x \leq y) \Longrightarrow x \leq y$ **by** *simp*
qed
from *H3 this* **have** $(\bigwedge y.\ y \in Y \Longrightarrow y \gtrsim X \Longrightarrow (if\ consi\ X\ Y\ then\ SOME$
$x.\ supRs\ x\ X\ Y\ else\ undefined) \leq y)$ **by**(*simp add*: *supRs-def*)
then show $(\bigwedge y.\ y \in Y \Longrightarrow y \gtrsim X \Longrightarrow SupFun\ X\ Y \leq y)$ **by**(*simp add*:
SupFun-def)
qed
qed

lemma *SupFun-is-sup*: $[\![$ *ccpo* $(Y :: ('a :: order)\ set)$; $(X :: 'a\ set) \subseteq Y$;
consi $X\ Y\ ]\!] \Longrightarrow supRs\ (SupFun\ X\ Y)\ X\ Y$
by (*simp add*: *SupFun-greater SupFun-least SupFun-inX supRs-def*)

abbreviation $pot\ X \equiv \{f :: ('d \Rightarrow 'a).\ range\ f \subseteq X\}$

lemma *function-space-ccpo*:
$[\![$ *ccpo* $(X :: ('a :: order)\ set)]\!] \Longrightarrow ccpo\ (pot\ X)$
proof$-$
fix $X :: 'a\ set$
assume *Hccpo*: *ccpo X*
then have *H1*: $\forall X'{\subseteq}X.\ consi\ X'\ X \longrightarrow (\exists b{\in}X.\ supRs\ b\ X'\ X)$ **by** (*simp add*:
ccpo-def)
have $\forall\ F \subseteq\ pot\ X.\ consi\ F\ (pot\ X) \longrightarrow (\exists\ f \in\ pot\ X.\ supRs\ f\ F\ (pot\ X)\ )$
proof
fix $F :: ('d \Rightarrow 'a)\ set$
show $F \subseteq pot\ X \longrightarrow consi\ F\ (pot\ X) \longrightarrow (\exists f{\in}pot\ X.\ supRs\ f\ F\ (pot\ X))$
proof
assume *H2*: $F \subseteq pot\ X$
show $consi\ F\ (pot\ X) \longrightarrow (\exists f{\in}pot\ X.\ supRs\ f\ F\ (pot\ X))$ **proof**
assume *H3*: $consi\ F\ (pot\ X)$

let *?fd* = ‹($\lambda d.$ *SupFun* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X)›
show ($\exists f{\in}$*pot* X. *supRs f F* (*pot X*)) **proof**

have *Sconsid*: $\bigwedge$ *d*. *consi* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X **proof** −
fix *d* :: *'d*
have $\bigwedge x\ y.\ (\exists f{\in}F.\ x = f\ d) \Longrightarrow (\exists f{\in}F.\ y = f\ d) \Longrightarrow (\exists\ b{\in}X.\ x \le b \wedge y \le b)$ **proof**−
fix *x*
fix *y*
assume $(\exists f{\in}F.\ x = f\ d)$
then obtain *fx* **where** *Hfx1*: *fx* ∈ *F* **and** *Hfx2*: *x* = *fx d* **by** *auto*
assume $(\exists f{\in}F.\ y = f\ d)$
then obtain *fy* **where** *Hfy1*: *fy* ∈ *F* **and** *Hfy2*: *y* = *fy d* **by** *auto*
from *H3 Hfx1 Hfy1 consi-def*[*of F pot X*] **have** ∃ *fbound*. *range fbound* ⊆ *X* ∧ *fx* ≤ *fbound* ∧ *fy* ≤ *fbound* **by** *simp*
then obtain *fbound* **where** *Hfbound1*: *range fbound* ⊆ *X* **and** *Hfbound2*: *fx* ≤ *fbound* **and** *Hfbound3*: *fy* ≤ *fbound* **by** *auto*
show $(\exists\ b{\in}X.\ x \le b \wedge y \le b)$ **proof**
from *Hfbound2 Hfbound3* **have** *fx d* ≤ *fbound d* ∧ *fy d* ≤ *fbound d*
by (*simp add*: *le-funD*)
then show *x* ≤ *fbound d* ∧ *y* ≤ *fbound d* **by**(*simp add*: *Hfx2 Hfy2*)
next
from *Hfbound1* **show** *fbound d* ∈ *X* **by** *auto*
qed
qed
then show *consi* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X **by**(*simp add*: *consi-def*)
qed

have *SsupRs*:$\bigwedge$ *d*. *supRs* (*SupFun* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X) $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X **proof** −
fix *d* :: *'d*
from *Sconsid*[*of d*] *Hccpo SupFun-is-sup*[*of X* $\{y.\ \exists f{\in}F.\ y = f\ d\}$]
show *supRs* (*SupFun* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X) $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X
using *H2* **by** *blast*
qed

show *supRs ?fd F* (*pot X*) **proof**−
have (*?fd* ∈ *pot X*) ∧ ($\forall f{\in}F.\ f \le (\lambda d.$ *SupFun* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X)) ∧
($\forall f.$ *range f* ⊆ *X* ⟶ ($\forall\,ya{\in}F.\ ya \le f$) ⟶ ($\lambda d.$ *SupFun* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X) ≤ *f*) **proof**
show *?fd* ∈ *pot X*
by (*metis* (*no-types*, *lifting*) *SsupRs image-subset-iff mem-Collect-eq supRs-def*)
next
show ($\forall f{\in}F.\ f \le (\lambda d.$ *SupFun* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X)) ∧
($\forall f.$ *range f* ⊆ *X* ⟶ ($\forall\,ya{\in}F.\ ya \le f$) ⟶ ($\lambda d.$ *SupFun* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X) ≤ *f*) **proof**
show $\forall f{\in}F.\ f \le (\lambda d.$ *SupFun* $\{y.\ \exists f{\in}F.\ y = f\ d\}$ X) **proof**

fix f
assume *Hf*: $f \in F$
from *SsupRs* **have** $\bigwedge\ d.\ (SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X) \gtrsim \{y.\ \exists f{\in}F.\ y = f\ d\}$ **by**(*simp add*: *supRs-def*)
from *this Hf* **have** $\bigwedge\ d.\ f\ d \le SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X$ **by** *auto*
from *this* **show** $f \le (\lambda d.\ SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X)$ **by** (*simp add*: *le-funI*)
qed
next
show $(\forall f.\ range\ f \subseteq X \longrightarrow (\forall ya{\in}F.\ ya \le f) \longrightarrow (\lambda d.\ SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X) \le f)$ **proof**
fix $f :: 'd \Rightarrow 'a$
show $(range\ f \subseteq X \longrightarrow (\forall ya{\in}F.\ ya \le f) \longrightarrow (\lambda d.\ SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X) \le f)$ **proof**
assume *Hf1*: $range\ f \subseteq X$
show $(\forall ya{\in}F.\ ya \le f) \longrightarrow (\lambda d.\ SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X) \le f$ **proof**
assume *Hf2*: $(\forall ya{\in}F.\ ya \le f)$
have $\bigwedge\ d.\ (SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X) \le (f\ d)$ **proof**−
fix $d :: 'd$
from *Hf1 Hf2 SsupRs*[*of d*] *supRs-def*[*of* $(SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X)\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X$]
show $(SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X) \le f\ d$
by (*smt le-funD mem-Collect-eq rangeI subsetCE*)
qed
from *this* **show** $(\lambda d.\ SupFun\ \{y.\ \exists f{\in}F.\ y = f\ d\}\ X) \le f$ **by**(*simp add*: *le-funI*)
qed
qed
qed
qed
qed
from *this* **show** *?thesis* **by**(*simp add*: *supRs-def*)
qed

have *fdprop*: $?fd \in pot\ X$
by (*metis* (*no-types*, *lifting*) *SsupRs image-subset-iff mem-Collect-eq supRs-def*)

from *this* **have** $\forall\ d :: 'd.\ ?fd(d) \in X$ **by** *auto*
from *this supRs-def* **show** $?fd \in pot\ X$ **by** *auto*
qed
qed
qed
qed
then show *ccpo* (*pot X*) **by** (*simp add*:*ccpo-def*)
qed

lemma *function-space-ccpo-full*:
ccpo (*UNIV* :: (('*a* ::*order*) *set*)) $\Longrightarrow$ *ccpo* (*UNIV* :: ('*d* $\Rightarrow$ '*a*) *set*)
using *function-space-ccpo* **by** *fastforce*

lemma *function-space-ccpo-bool3*:
ccpo (*UNIV* :: ('*d* $\Rightarrow$ *bool3*) *set*)
by (*simp add*: *function-space-ccpo-full UNIV-bool3 b3-ccpo*)

lemma *function-space-ccpo-bool4*:
ccpo (*UNIV* :: ('*d* $\Rightarrow$ *bool4*) *set*)
by (*simp add*: *function-space-ccpo-full UNIV-bool4 b4-ccpo*)

lemma *function-space-complete-lattice*:
$[\![$ *complete-latticeR* (*X* :: ('*a* :: *order*) *set*)$]\!]$ $\Longrightarrow$ *complete-latticeR* (*pot X*)
proof−
fix *X* :: ('*a* :: *order*) *set*
assume *H*: *complete-latticeR X*
then have $\exists$ *x* $\in$ *X*. *infRs x* {} *X* **by**(*simp add*:*complete-latticeR-def*)
then obtain *x* **where** *Hx1*: *x* $\in$ *X* **and** *Hx2*: *infRs x* {} *X* **by** *auto*
from *Hx2* **have** *Hxupperbound*: *x* $\gtrsim$ *X* **by**(*simp add*: *infRs-def*)
let *?fx* = (λ *d*. *x*)
from *Hxupperbound* **have** *Hfxupperbound*: *?fx* $\gtrsim$ *pot X* **using** *le-fun-def* **by** *fastforce*

from *Hx1* **have** *Hfx1*: *?fx* $\in$ (*pot X*) **by** *auto*

from *H* **have** *ccpo X* **by**(*simp add*: *ccpo-def complete-latticeR-def*)
then have *ccpo* (*pot X*) **using** *function-space-ccpo* **by** *auto*
from *this H Hfxupperbound Hfx1* **show** *complete-latticeR* (*pot X*)
using *greatest-element-implies-complete-lattice*[*of pot X*] **by** *blast*
qed

lemma *function-space-complete-lattice-full*:
complete-latticeR (*UNIV* :: ('*a* :: *order*) *set*) $\Longrightarrow$ *complete-latticeR* (*UNIV* :: ('*d* $\Rightarrow$ '*a*) *set*)
using *function-space-complete-lattice* **by** *fastforce*

lemma *function-space-complete-lattice-bool4*:
complete-latticeR (*UNIV* :: ('*d* $\Rightarrow$ *bool4*) *set*)
using *UNIV-bool4 b4-complete-lattice*
by (*simp add*: *function-space-complete-lattice-full insert-commute*)

definition *soundp*:: ('*a* :: *order*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool* **where**
soundp x f $\equiv$ (*x* $\leq$ *f x*)

definition *repletep*:: ('*a* :: *order*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool* **where**
repletep x f $\equiv$ (*f x* $\leq$ *x*)

definition *fixedp* :: ('*a* :: *order*) $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool* **where**

fixedp x f $\equiv$ ($x = f\ x$)

abbreviation *monot* **where**
monot (f :: (('a ::*order*) $\Rightarrow$ 'a)) $\equiv$
($\forall\ a$ :: 'a. $\forall\ b$ :: 'a. $a \leq b \longrightarrow f\ a \leq f\ b$)

lemma *max-elem-in-ccpo*:
$[\![$ *ccpo* (X :: ('a :: *order*) *set*); $x \in X$ $]\!]$
$\Longrightarrow \exists\ m \in X.$ ($\forall\ a \in X.\ m \leq a \longrightarrow m = a) \wedge x \leq m$
proof−
fix X :: ('a ::*order*) *set*
fix x :: 'a
assume *H1*: *ccpo X*
assume *H2*: $x \in X$
let *?Y* = $\{y \in X.\ x \leq y\}$
from *H1 H2* **have** *H3*: *ccpo ?Y* **using** *above-sublattice-is-ccpo* **by** *auto*

let *?r* = { ((a :: 'a), (b::'a)) . $a \in$ *?Y* $\wedge$ $b \in$ *?Y* $\wedge$ $a \leq b$}
have $x \leq x$ **by** *auto*
from *this* **have** *Hf*: *Field ?r* = *?Y* **by**(*simp add*: *Field-def*; *auto*)
have *Hr*: *refl-on ?Y ?r* **by** (*simp add*: *refl-on-def'*)
have *Ha*: *antisym ?r* **by** (*simp add*: *antisym-def*; *auto*)
have *Ht*: *trans ?r* **by** (*simp add*: *trans-def*; *auto*)
from *Hf Hr Ha Ht* **have** *Hpo*: *Partial-order ?r*
by (*simp add*: *partial-order-on-def preorder-on-def*)

have $\forall$ *C*$\in$*Chains ?r*. $\exists$ *u*$\in$*Field ?r*. $\forall$ *a*$\in$*C*. (a, u) $\in$ *?r* **proof**
fix C :: 'a *set*
assume *HC*: $C \in$ *Chains ?r*
from *this* **have** *HCss*: $C \subseteq$ *?Y* **by** (*simp add*: *Chains-def*; *auto*)

from *HC* **have** $\forall\ a \in C$. $\forall\ b \in C.\ a \leq b \vee b \leq a$ **by**(*simp add*: *Chains-def*; *auto*)
from *this* **have** $\forall\ a \in C$. $\forall\ b \in C.\ \exists\ u \in C.\ a \leq u \wedge b \leq u$ **by** *auto*
from *HCss this consi-def*[*of C ?Y*] **have** *consi C ?Y* **by** *blast*
from *this H3 ccpo-def*[*of ?Y*] **have** ($\exists$ *b*$\in$*?Y*. *supRs b C ?Y*) **using** *HCss* **by** *blast*
then obtain *b* **where** *Hb1*: *b*$\in$ *?Y* **and** *Hb2*: *supRs b C ?Y* **by** *auto*
from *Hb2 supRs-def* **have** $b \gtrsim C$ **by** *auto*

from *Hb1 this* **have** $\exists\ u \in$ *?Y*. $u \gtrsim C$ **by** *auto*
from *this Hf HCss* **show** $\exists$ *u*$\in$*Field ?r*. $\forall$ *a*$\in$*C*. (a, u) $\in$ *?r* **by** *auto*
qed

from *Hf Hpo Zorns-po-lemma*[*of ?r*] *this* **have** $\exists$ *m*$\in$*?Y*. $\forall$ *a*$\in$*?Y*. $m \leq a \longrightarrow a = m$ **by** *simp*
from *this* **show** $\exists\ m \in X.$ ($\forall\ a \in X.\ m \leq a \longrightarrow m = a) \wedge x \leq m$
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq order-trans*)
qed

lemma *soundp-implies-fixedp*:
$\llbracket$ *ccpo* (*X* :: ('*a* ::*order*) *set*); $x \in X$; *soundp x f*; *monot* (*f* :: ('*a* ⇒ '*a*)); *f*'*X* ⊆ *X* $\rrbracket$
  $\Longrightarrow \exists\ z \in X.\ x \leq z \wedge$ *fixedp z f*
proof−
 fix *X* :: '*a set* **fix** *x* :: '*a*
 assume *H1*: *ccpo X* **assume** *H2*: $x \in X$
 fix *f*:: '*a* ⇒ '*a*
 assume *H3*: *soundp x f* **assume** *H4*: *monot f* **assume** *Himgf*: *f*'*X* ⊆ *X*

 let *?Y* = { $y \in X$. *soundp y f* ∧ $x \leq y$}
 have *ccpo ?Y* **proof**−
  have ∀ *X*'⊆*?Y*. *consi X' ?Y* ⟶ (∃ *b*. *b* ∈ *?Y* ∧ *supRs b X' ?Y*) **proof**
   fix *Z* :: '*a set*
   show *Z*⊆*?Y* ⟶ *consi Z ?Y* ⟶ (∃ *b*. *b* ∈ *?Y* ∧ *supRs b Z ?Y*) **proof**
    assume *H5*: *Z*⊆*?Y*
    show *consi Z ?Y* ⟶ (∃ *b*. *b* ∈ *?Y* ∧ *supRs b Z ?Y*) **proof**(*cases Z*={})
     case *True*
     have (∃ *b*. *b* ∈ *?Y* ∧ *supRs b Z ?Y*) **proof**
      have *x* ∈ *?Y* ∧ *supRs x* {} *?Y* **by** (*simp add*: *H2 H3 supRs-def*)
      from *this True* **show** *x* ∈ *?Y* ∧ *supRs x Z ?Y* **by** *auto*
     qed
     from *this* **show** *?thesis* **by** *auto*
    next
     case *False*
     show *consi Z ?Y* ⟶ (∃ *b*. *b* ∈ *?Y* ∧ *supRs b Z ?Y*) **proof**
     assume *H6*: *consi Z ?Y*
      have *HYssX*: *?Y* ⊆ *X* **by** *blast*
      from *this H6* **have** *Hzconsi*: *consi Z X* **using** *consi-subset* **by** *blast*
      from *HYssX* **and** *H5* **have** *Z* ⊆ *X* **by** *auto*
       from *this Hzconsi H1 ccpo-def*[*of X*] **have** (∃ *b*∈*X*. *supRs b Z X*) **by** *auto*
      then obtain *b* **where** *Hb1*: *b* ∈ *X* **and** *Hb2*: *supRs b Z X* **by** *auto*
      from *Hb2 H2* **have** *HbgrZ*: *b* ≳ *Z* **by** (*simp add*: *supRs-def*)
      from *this False* **have** *Hbgreaterx*: *b* ≥ *x*
          by (*metis* (*no-types*, *lifting*) *Ball-Collect H5 atLeastAtMost-iff atLeastatMost-empty-iff bot.extremum-uniqueI empty-iff subset-emptyI*)
      have *Hbsound*: *soundp b f* **proof**−
       have *Hfbub*: *f b* ≳ *Z* **proof**
        fix *z*
        assume *HzinZ*:*z* ∈ *Z*
        from *this* **have** *z* ≤ *b* **using** *HbgrZ* **by** *blast*
        from *this H4* **have** *Hord*: *f z* ≤ *f b* **by** *auto*
        from *HzinZ H5 soundp-def*[*of z f*] **have** *z* ≤ *f z* **by** *blast*
        from *this Hord* **show** *z* ≤ *f b* **by** *auto*
       qed
       from *Himgf H2* **have** *HbinX*: *f b* ∈ *X*
        using *Hb1* **by** *blast*

from *Hb2* **have** $(\forall\ y' \in X.\ y' \gtrsim Z \longrightarrow b \leq y')$ **by** (*simp add*: *supRs-def*)
from *this Hfbub HbinX* **have** $b \leq f\ b$ **by** *simp*
from *this* **show** *?thesis* **by**(*simp add*: *soundp-def*)
qed
from *Hb1 Hbgreaterx Hbsound* **have** *binY*: $b \in ?Y$ **by** *auto*

have *bisSup*: *supRs b Z ?Y* **proof**−
have *Hsupb1*: $b \in ?Y$ **using** *binY* **by** *auto*
have *Hsupb2*: $b \gtrsim Z$ **using** *HbgrZ* **by** *auto*
from *Hb2* **have** $\forall\ y \in X.\ y \gtrsim Z \longrightarrow b \leq y$ **by** (*simp add*: *supRs-def*)
from *this* **have** *Hsupb3*: $\forall\ y \in ?Y.\ y \gtrsim Z \longrightarrow b \leq y$ **by** *simp*
from *Hsupb1 Hsupb2 Hsupb3 supRs-def*[*of b Z ?Y*] **show** *?thesis* **by**(*simp*)
qed
show $(\exists\, b.\ b \in ?Y \wedge$ *supRs b Z ?Y*$)$ **proof**
from *binY bisSup* **show** $b \in ?Y \wedge$ *supRs b Z ?Y* **by** *auto*
qed
qed
qed
qed
qed
then show *?thesis* **by** (*simp add*:*ccpo-def*)
qed

from *this max-elem-in-ccpo*[*of ?Y x*] **have**
$\exists\ m \in ?Y.\ (\forall\ a \in ?Y.\ m \leq a \longrightarrow m = a) \wedge x \leq m$
using *H2 H3* **by** *blast*
from *this* **obtain** *m* **where** *Hm1*: $m \in ?Y$ **and** *Hm2*: $(\forall\ a \in ?Y.\ m \leq a \longrightarrow m = a)$
and *Hm3*: $x \leq m$ **by** *auto*
from *Hm1* **have** $m \leq f\ m$ **by**(*simp add*: *soundp-def*)
from *this H4* **have** $f\ m \leq f\ (f\ m)$ **by**(*simp*)
then have *Hfsp*: *soundp* $(f\ m)$ *f* **by**(*simp add*: *soundp-def*)
from *Hm1 Himgf* **have** *HfminX*: $(f\ m\ ) \in X$ **by** *auto*
from *Hm3* ‹$m \leq f\ m$› **have** $x \leq f\ m$ **by** *auto*
from *Hfsp this HfminX* **have** $(f\ m) \in ?Y$ **by** *auto*

from *this Hm2* ‹$m \leq f\ m$› **have** $(f\ m) = m$ **by** *auto*
from *this* **have** *fixedp m f* **by**(*simp add*: *fixedp-def*)

from *Hm1 this Hm3* **show** $\exists\ z \in X.\ x \leq z \wedge$ *fixedp z f* **by** *auto*
qed

lemma *soundp-repletep-implies-fixedp*:
$[\![$ *ccpo* $(X :: ('a :: order)\ set)$; $x \in X$; $y \in X$; *soundp x f*;
repletep y f; $x \leq y$; *monot* $(f :: ('a \Rightarrow 'a))$; $f`X \subseteq X\ ]\!]$
$\Longrightarrow \exists\ z \in X.\ x \leq z \wedge z \leq y \wedge$ *fixedp z f*
proof−

fix X :: $('a$ ::*order*$)$ *set* **fix** x :: $'a$ **fix** y :: $'a$
assume *H1*: *ccpo* X **assume** *H2*: $x \in X$ **assume** *H3*: $y \in X$
fix f:: $'a \Rightarrow 'a$
assume *H4*: *soundp* $x\ f$ **assume** *H5*: *repletep* $y\ f$
assume *H6*: *monot* f **assume** *Himgf*: $f`X \subseteq X$
assume *Hxy*: $x \leq y$

let $?ZA = \langle\{z \in X.\ x \leq z\}\rangle$
from *H1 H2* **have** *ccpo ?ZA* **using** *above-sublattice-is-ccpo* **by** *auto*

let $?ZB = \langle\{z \in ?ZA.\ z \leq y\}\ \rangle$
from ⟨*ccpo ?ZA*⟩ *H1 H3 Hxy* **have** *complete-latticeR ?ZB*
using *below-sublattice-is-complete* **by** *blast*
then have *ccpo ?ZB* **using** *complete-lattice-implies-ccpo* **by** *auto*
let $?Z = \{z \in X.\ x \leq z \wedge z \leq y\}$
from ⟨*ccpo ?ZB*⟩ **have** *ccpo ?Z* **by** *simp*
from *H2 Hxy* **have** $x \in ?Z$ **by** *simp*

have $f`\ ?Z \subseteq ?Z$ **proof**
fix z'
assume $z' \in f`\ ?Z$
from *this* **have** $\exists\ x' \in ?Z.\ z' = f\ x'$ **by** *auto*
then obtain x' **where** *Hx'1*: $x' \in ?Z$ **and** *Hx'2*:$z' = f\ x'$ **by** *auto*

from *Hx'1* **have** $x \leq x'$ **and** $x' \leq y$ **by** *auto*
from *H6* **have** $f\ x \leq f\ x'$ **using** $\langle x \leq x'\rangle$ **by** *simp*
from *H6* **have** $f\ x' \leq f\ y$ **using** $\langle x' \leq y\rangle$ **by** *simp*

from *H4 soundp-def*[*of* x] $\langle f\ x \leq f\ x'\rangle$ **have** $\langle x \leq f\ x'\rangle$ **by** *auto*
from *H5 repletep-def*[*of* y] $\langle f\ x' \leq f\ y\rangle$ **have** $\langle f\ x' \leq y\rangle$ **by** *auto*
from *Hx'1 Himgf* **have** $\langle f\ x' \in X\rangle$ **by** *auto*
from $\langle x \leq f\ x'\rangle$ $\langle f\ x' \leq y\rangle$ $\langle f\ x' \in X\rangle$ **have** $\langle f\ x' \in ?Z\rangle$ **by** *simp*
from *this Hx'2* **show** $z' \in ?Z$ **by** *auto*
qed

from *this* ⟨*ccpo ?Z*⟩ $\langle x \in ?Z\rangle$ *soundp-implies-fixedp*[*of ?Z x f*] *H4 H6*
have $\exists\ p \in ?Z.$ *fixedp* $p\ f \wedge x \leq p$ **by** *simp*
then show ⟨ $\exists\ z \in X.\ x \leq z \wedge z \leq y \wedge$ *fixedp* $z\ f$⟩ **by** *auto*
qed

lemma *repletep-implies-fixedp*:
$[\![$ *ccpo* $(X$ :: $('a$ ::*order*$)$ *set*$)$; $x \in X$; *repletep* $x\ f$;
monot $(f :: ('a \Rightarrow 'a))$; $f`X \subseteq X$ $]\!]$
$\Longrightarrow \exists\ z \in X.\ z \leq x \wedge$ *fixedp* $z\ f$
proof−
fix X :: $'a$ *set* **fix** x :: $'a$
assume *H1*: *ccpo* X **assume** *H2*: $x \in X$
fix f:: $'a \Rightarrow 'a$
assume *H3*: *repletep* $x\ f$ **assume** *H4*: *monot* f **assume** *Himgf*: $f`X \subseteq X$

from *H1* **have** $\exists\ b \in X.\ b \lesssim X$ **using** *ccpo-least-element* **by** *auto*
then obtain *b* **where** *Hb1*: $b \in X$ **and** *Hb2*: $b \lesssim X$ **by** *auto*
from *H2 Hb2* **have** ‹$b \le x$› **by** *auto*
from *Hb2 Himgf* **have** $b \le f\ b$ **using** *Hb1* **by** *blast*
then have *soundp b f* **by** (*simp add*: *soundp-def*)
from *this soundp-repletep-implies-fixedp*[*of X b x f*]
show $\exists\ z \in X.\ z \le x \wedge$ *fixedp z f*
using *H1 H2 H3 H4 Hb1 Himgf* ‹$b \le x$› **by** *blast*
qed

definition *FixPs* ::‹('*a* ::*order*) *set* ⇒ ('*a* ⇒ '*a*) ⇒ '*a set*›
where *FixPs X f* = $\{x \in X.\ f(x) = x\}$

lemma *VisserFixp*:
⟦ *ccpo* (*X*:: ('*a*::*order*) *set*); *monot* (*f* :: '*a* ⇒ '*a*); *f*‘*X* ⊆ *X* ⟧
⟹ *ccpo* (*FixPs X f*)
proof−
fix *X* :: ('*a* :: *order*) *set*
fix *f* :: '*a* ⇒ '*a*
assume *Hccpo*: *ccpo X* **assume** *Hmon*: *monot f*
assume *Himgf*: *f*‘*X* ⊆ *X*
show *ccpo* (*FixPs X f*) **proof**−
have ∀ *Y* ⊆ (*FixPs X f*). *consi Y* (*FixPs X f*)
⟶ (∃ *b* ∈ (*FixPs X f*). *supRs b Y* (*FixPs X f*)) **proof**
fix *Y*
show *Y* ⊆ (*FixPs X f*) ⟶ *consi Y* (*FixPs X f*) ⟶ (∃ *b* ∈ (*FixPs X f*). *supRs b Y* (*FixPs X f*)) **proof**
assume ‹*Y* ⊆ (*FixPs X f*)›
show *consi Y* (*FixPs X f*) ⟶ (∃ *b* ∈ (*FixPs X f*). *supRs b Y* (*FixPs X f*)) **proof**
assume ‹*consi Y* (*FixPs X f*)›
have (*FixPs X f*) ⊆ *X* **using** *FixPs-def* **by** *auto*
from *this* ‹*consi Y* (*FixPs X f*)› **have** ‹*consi Y X*› **using** *consi-subset* **by** *auto*
from *this Hccpo ccpo-def*[*of X*] **have** (∃ *b*∈*X*. *supRs b Y X*)
using ‹*Y* ⊆ *FixPs X f*› ‹*FixPs X f* ⊆ *X*› **by** *auto*
then obtain *b* **where** ‹*b* ∈ *X*› **and** ‹*supRs b Y X*› **by** *auto*
from ‹*supRs b Y X*› **have** ‹∀ *y* ∈ *Y*. $y \le b$› **using** *supRs-def* **by** *auto*
from ‹*Y* ⊆ (*FixPs X f*)› **have** ‹∀ *y* ∈ *Y*. *y* = *f y*›
by (*simp add*: *FixPs-def subset-eq*)
from *this* ‹∀ *y* ∈ *Y*. $y \le b$› **have** ‹$f\ b \gtrsim Y$› **using** *Hmon* **by** *fastforce*
from ‹*b* ∈ *X*› *Himgf* **have** ‹*f b* ∈ *X*› **by** *auto*
from *this* ‹$f\ b \gtrsim Y$› ‹*supRs b Y X*› **have** ‹$b \le f\ b$› **by** (*simp add*: *supRs-def*)
from *this* **have** ‹*soundp b f*› **using** *soundp-def* **by** *auto*

let *?Z* = $\{z \in X.\ f(z) = z \wedge b \le z\}$
from ‹*soundp b f*› *soundp-implies-fixedp*[*of X b f*]
fixedp-def **have** ‹*?Z* ≠ {}› **using** *Hccpo* ‹*b* ∈ *X*› *Hmon Himgf*
by (*metis* (*mono-tags*, *lifting*) *empty-iff mem-Collect-eq*)
from *Hccpo this ccpo-nonempty-has-inf*[*of X ?Z*]

```
    have ∃ i∈ X. infRs i ?Z X by auto
    from this obtain i where Hi1: i ∈ X and Hi2: ‹infRs i ?Z X› by auto

    have ‹∀ z ∈ ?Z. f z = z› by auto
    from Hi2 have ∀ z ∈ ?Z. i ≤ z by(simp add: infRs-def)
    from this Hmon have ∀ z ∈ ?Z. f i ≤ f z by blast
    from this ‹∀ z ∈ ?Z. f z = z› have ‹f i ≲ ?Z›
      by (metis (no-types, lifting))
    from Hi1 Himgf have ‹f i ∈ X› by auto
    from ‹f i ∈ X› ‹f i ≲ ?Z› Hi2 have ‹f i ≤ i› by(simp add: infRs-def)
    from this have ‹repletep i f› by(simp add: repletep-def)

    have ‹b ≲ ?Z› by auto
    from ‹b ∈ X› this Hi2 have ‹b ≤ i› by(simp add: infRs-def)

    from this ‹repletep i f› ‹soundp b f› Hi1 ‹b ∈ X› Hmon Himgf Hccpo
   have ‹∃ z∈X. b ≤ z ∧ z ≤ i ∧ fixedp z f› using soundp-repletep-implies-fixedp[of
X b i f]
      by auto
    then obtain b' where ‹b' ∈ X› and ‹b ≤ b'› and ‹b' ≤ i› and ‹fixedp b' f›
      by auto
    from ‹fixedp b' f› have ‹f b' = b'› by(simp add: fixedp-def)
    from this ‹b ≤ b'› ‹b' ∈ X› have b' ∈ ?Z by auto
    from ‹supRs b Y X› ‹b ≤ b'› have ‹b' ≳ Y› by(simp add: supRs-def) auto
    from Hi2 ‹b' ≤ i› have ‹b' ≲ ?Z› by(simp add: infRs-def) auto

    from ‹fixedp b' f› have ‹f b' = b'› by(simp add: fixedp-def)
    then have ‹b' ∈ (FixPs X f)› using ‹b' ∈ X› by(simp add: FixPs-def)
    have ∀ y∈ (FixPs X f). ( y ≳ Y ⟶ b' ≤ y ) proof
        fix y
        assume ‹y ∈ FixPs X f›
        from this FixPs-def have ‹f y = y› by auto
        from ‹y ∈ FixPs X f› FixPs-def have ‹ y ∈ X› by auto
        show  y ≳ Y ⟶ b' ≤ y  proof
          assume ‹y ≳ Y›
         from this ‹ y ∈ X› ‹supRs b Y X› have ‹b ≤ y› by(simp add: supRs-def)
          from this ‹f y = y› ‹ y ∈ X› have ‹y ∈ ?Z› by auto
          from ‹y ∈ ?Z› ‹b' ≲ ?Z› show ‹b' ≤ y› by auto
        qed
      qed

    from this ‹b' ≳ Y› ‹b' ∈ (FixPs X f)› supRs-def[of b' Y (FixPs X f)]
    have ‹supRs b' Y (FixPs X f)› by simp
    then show ( ∃ b ∈ (FixPs X f). supRs b Y (FixPs X f))
      using ‹b' ∈ (FixPs X f)› by auto
  qed
qed
qed
  from this show ?thesis by(simp add: ccpo-def)
```

qed
qed

lemma *VisserFixp2*:
⟦ *ccpo* (*X*:: (′*a*::*order*) *set*); *monot* (*f* :: ′*a* ⇒ ′*a*); *f*‘*X* ⊆ *X*;
∃ *g* ∈ *X*. *g* ≳ *X* ⟧ ⟹ ∃ *g*′ ∈ (*FixPs X f*). *g*′ ≳ (*FixPs X f*)
proof−
fix *X* :: (′*a* :: *order*) *set*
fix *f* :: ′*a* ⇒ ′*a*
assume *Hccpo*: *ccpo X* **assume** *Hmon*: *monot f*
assume *Himgf*: *f*‘*X* ⊆ *X* **assume** ‹∃ *g* ∈ *X*. *g* ≳ *X*›
then obtain *g* **where** *Hg1*: ‹*g* ∈ *X*› **and** *Hg2*: ‹*g* ≳ *X*› **by** *auto*
from *Hg1 Hg2* **have** *consi* (*FixPs X f*) *X*
by(*simp add*: *consi-def FixPs-def*) *auto*
from *ccpo-def*[*of X*] **have** ‹∃ *b* ∈ *X*. *supRs b* (*FixPs X f*) *X*›
by (*metis* (*no-types*, *lifting*) *FixPs-def Hccpo* ‹*consi* (*FixPs X f*) *X*› *mem-Collect-eq subsetI*)
then obtain *b* **where** *Hb1*: ‹*b* ∈ *X*› **and** *Hb2*: *supRs b* (*FixPs X f*) *X* **by** *auto*
from ‹*supRs b* (*FixPs X f*) *X*› **have** ‹∀ *y* ∈ (*FixPs X f*). *y* ≤ *b*› **using** *supRs-def* **by** *auto*
have ‹∀ *y* ∈ (*FixPs X f*). *y* = *f y*› **by** (*simp add*: *FixPs-def*)
from *this* ‹∀ *y* ∈ (*FixPs X f*). *y* ≤ *b*› **have** ‹*f b* ≳ (*FixPs X f*)›
using *Hmon* **by** *fastforce*
from ‹*b* ∈ *X*› *Himgf* **have** ‹*f b* ∈ *X*› **by** *auto*
from *this* ‹*f b* ≳ (*FixPs X f*)› ‹*supRs b* (*FixPs X f*) *X*› **have** ‹*b* ≤ *f b*› **by** (*simp add*: *supRs-def*)
then have ‹*soundp b f*› **using** *soundp-def* **by** *auto*
from *soundp-implies-fixedp*[*of X b f*] ‹*b* ∈ *X*› ‹*soundp b f*›
Hmon Himgf Hccpo **have** ∃ *z*∈*X*. *b* ≤ *z* ∧ *fixedp z f* **by** *auto*
from *this* **obtain** *z* **where** *Hz1*: ‹*z* ∈ *X*› **and** *Hz2*: ‹*b* ≤ *z*› **and** *Hz3*: ‹*fixedp z f*› **by** *auto*
from *Hz3 fixedp-def*[*of z f*] **have** (*f z*) = *z* **using** *sym* **by** *simp*
from *Hz1 this* **have** *z* ∈ {*x* ∈ *X*. (*f x*) = *x*} **by** *simp*
from *this FixPs-def*[*of X f*] **have** ‹*z* ∈ *FixPs X f*› **by** *auto*
show ∃ *g*′ ∈ (*FixPs X f*). *g*′ ≳ (*FixPs X f*) **proof**
show ‹*z* ∈ *FixPs X f*› **using** ‹*z* ∈ *FixPs X f*› **by** *auto*
next
show ∀ *y*∈(*FixPs X f*). *y* ≤ *z* **proof**
fix *y*
assume ‹ *y* ∈ (*FixPs X f*)›
from *this* ‹*supRs b* (*FixPs X f*) *X*› **have** ‹*y* ≤ *b*› **by**(*simp add*: *supRs-def*)
from ‹*b* ≤ *z*› ‹*y* ≤ *b*› **show** ‹*y* ≤ *z*› **by** *auto*
qed
qed
qed

lemma *KnasterTarski*:
⟦ *complete-latticeR* (*X*:: (′*a*::*order*) *set*); *monot* (*f* :: ′*a* ⇒ ′*a*); *f*‘*X* ⊆ *X* ⟧
⟹ *complete-latticeR* (*FixPs X f*)

proof−
  fix *X* :: (′*a* :: *order*) *set*
  fix *f* :: ′*a* ⇒ ′*a*
  assume *Hcompl*: *complete-latticeR X* **assume** *Hmon*: *monot f*
  assume *Himgf*: *f*‘*X* ⊆ *X*
  from *Hcompl complete-lattice-implies-ccpo* **have** ‹*ccpo X*› **by** *auto*
  from *this VisserFixp Hmon Himgf* **have** ‹*ccpo* (*FixPs X f*)› **by** *auto*
  from *Hcompl* **have** (∃ *i*∈*X*. *infRs i* {} *X*) **by** (*simp add*: *complete-latticeR-def*)
  then obtain *g* **where** *Hg1*: ‹*g* ∈ *X*› **and** *Hg2*:‹*infRs g* {} *X*› **by** *auto*
  from *this* **have** ‹*g* ≳ *X*› **by** (*simp add*: *infRs-def*)

  from *this Hmon Himgf* ‹*ccpo X*› *VisserFixp2 Hg1* **have** ∃ *g*′∈*FixPs X f*. *g*′ ≳ (*FixPs X f*)
    by *blast*

  from *this greatest-element-implies-complete-lattice*
  show *complete-latticeR* (*FixPs X f*) **using** ‹*ccpo* (*FixPs X f*)› **by** *blast*
qed

definition *IntrPs* ::‹(′*a* ::*order*) *set* ⇒ ′*a set*›
where *IntrPs X* = {*x*∈ *X*. ∀ *y* ∈ *X*. *consi* {*x*,*y*} *X*}

lemma *intrinsic-lattice*:
⟦ *ccpo* (*X*:: (′*a*::*order*) *set*) ⟧
    ⟹ *complete-latticeR* (*IntrPs X*)
proof−
  fix *X* ::‹ ′*a set*›
  assume ‹*ccpo X*›

  have ‹∀ *Y* ⊆ (*IntrPs X*). ∃ *b* ∈ (*IntrPs X*). *supRs b Y* (*IntrPs X*)› **proof**
    fix *Y* :: ‹ ′*a set*›
    show ‹*Y* ⊆ (*IntrPs X*) ⟶ (∃ *b* ∈ (*IntrPs X*). *supRs b Y* (*IntrPs X*))› **proof**
      assume ‹*Y* ⊆ (*IntrPs X*)›
      then have ‹*Y* ⊆ *X*› **using** *IntrPs-def* **by** *auto*

      have ‹*consi Y X*› **proof**−
        have ∀ *x* ∈ *Y*. ∀ *y* ∈ *Y*. ∃ *b* ∈ *X*. *x* ≤ *b* ∧ *y* ≤ *b* **proof**
          fix *x* **assume** ‹*x* ∈ *Y*› **show** ∀ *y* ∈ *Y*. ∃ *b* ∈ *X*. *x* ≤ *b* ∧ *y* ≤ *b* **proof**
            fix *y* **assume** ‹*y* ∈ *Y*›
            from ‹*Y* ⊆ (*IntrPs X*)› ‹*x* ∈ *Y*› **have** ‹*x* ∈ (*IntrPs X*)› **by** *auto*
              from ‹*y* ∈ *Y*› ‹*Y* ⊆ *X*› **have** ‹*y* ∈ *X*› **by** *auto*
                from *this* ‹*x* ∈ (*IntrPs X*)› **have** ‹*consi* {*x*,*y*} *X*› **by**(*simp add*: *IntrPs-def*)
              from ‹*consi* {*x*,*y*} *X*› ‹*ccpo X*› *ccpo-def*
                obtain *b* **where** ‹*b* ∈ *X*› **and** ‹*supRs b* {*x*,*y*} *X*›
                  by (*metis* ‹*Y* ⊆ *X*› ‹*x* ∈ *Y*› ‹*y* ∈ *X*› *empty-subsetI insert-subset subsetCE*)
            show ∃ *b* ∈ *X*. *x* ≤ *b* ∧ *y* ≤ *b* **proof**
              show *b* ∈ *X* **using** ‹*b* ∈ *X*› **by** *auto*

```
          next
          show x≤ b ∧ y ≤ b using ‹supRs b {x,y} X› by (simp add: supRs-def)
          qed
        qed
      qed
      from this show ?thesis by(simp add: consi-def)
    qed
    from this ‹ccpo X› ‹Y ⊆ X› obtain b where ‹b ∈ X› and ‹supRs b Y X›
using ccpo-def
      by (blast)

    have ‹∀ x ∈ X. consi {b, x} X› proof
      fix x assume ‹x ∈ X›

      have (∀ x'∈Y ∪ {x}. ∀ y'∈Y ∪ {x}. ∃ b∈X. x' ≤ b ∧ y' ≤ b) proof
        fix x' assume ‹x' ∈ Y ∪ {x}›
        show ∀ y'∈Y ∪ {x}. ∃ b∈X. x' ≤ b ∧ y' ≤ b proof
          fix y' assume ‹y' ∈ Y ∪ {x}›
          show ∃ b∈X. x' ≤ b ∧ y' ≤ b proof(cases ‹y' ∈ Y›)
            case True
            from True ‹Y ⊆ (IntrPs X)› have ‹y' ∈ (IntrPs X)› by auto
           from ‹x' ∈ Y ∪ {x}› ‹Y ⊆ (IntrPs X)› ‹x ∈ X› IntrPs-def[of X] have
‹x' ∈ X› by auto
            from this ‹y' ∈ (IntrPs X)› IntrPs-def[of X] have ‹consi {x', y'} X›
              by (simp add: insert-commute)
            from True ‹Y ⊆ (IntrPs X)› IntrPs-def[of X] have ‹y' ∈ X› by auto
            from ‹consi {x', y'} X› ‹ccpo X› ‹x' ∈ X› ‹y' ∈ X› ccpo-def[of X]
              have ‹∃ b∈X. supRs b {x', y'} X› by simp
              then obtain b' where ‹b' ∈ X› and ‹supRs b' {x', y'} X› by auto
              show ?thesis proof
                show ‹b' ∈ X› using ‹b' ∈ X› by auto
                show ‹x' ≤ b' ∧ y' ≤ b'› using ‹supRs b' {x', y'} X›
                  supRs-def[of b' {x', y'} X] by simp
              qed
          next
            case False
            from this ‹y' ∈ Y ∪ {x}› have ‹y' = x› by auto
            then show ?thesis proof(cases ‹x' ∈ Y›)
              case True
              from True ‹Y ⊆ (IntrPs X)› have ‹x' ∈ (IntrPs X)› by auto
                from ‹x' ∈ Y ∪ {x}› ‹Y ⊆ (IntrPs X)› ‹x ∈ X› IntrPs-def[of X]
have ‹x' ∈ X› by auto
              from True ‹y' =x › ‹x ∈ X› have ‹y' ∈ X› by auto
             from this ‹x' ∈ (IntrPs X)› IntrPs-def[of X] have ‹consi {x', y'} X›
                by simp
              from ‹consi {x', y'} X› ‹ccpo X› ‹x' ∈ X› ‹y' ∈ X› ccpo-def[of X]
               have ‹∃ b∈X. supRs b {x', y'} X› by simp
               then obtain b' where ‹b' ∈ X› and ‹supRs b' {x', y'} X› by auto
               show ?thesis proof
```

show ‹$b' \in X$› **using** ‹$b' \in X$› **by** *auto*
show ‹$x' \leq b' \wedge y' \leq b'$› **using** ‹*supRs* b' $\{x', y'\}$ X›
supRs-def[*of* b' $\{x', y'\}$ X] **by** *simp*
qed
next
case *False*
from *this* ‹$x' \in Y \cup \{x\}$› **have** ‹$x' = x$› **by** *auto*
from ‹$x' = x$› ‹$y' = x$› ‹$x \in X$› **show** *?thesis* **by** *auto*
qed
qed
qed
qed
from *this* **have** ‹*consi* $(Y \cup \{x\})$ X› **by**(*simp add*: *consi-def*)
from ‹$x \in X$› ‹$Y \subseteq$ (*IntrPs* X)› *IntrPs-def* **have** ‹$(Y \cup \{x\}) \subseteq X$› **by** *blast*
from *this* ‹*ccpo* X› *ccpo-def* ‹*consi* $(Y \cup \{x\})$ X› **have** ‹$\exists$ $b \in X$. *supRs* b $(Y \cup \{x\})$ X› **by** *blast*
then obtain z **where** ‹$z \in X$› **and** ‹*supRs* z $(Y \cup \{x\})$ X › **by** *auto*
from ‹*supRs* z $(Y \cup \{x\})$ X › **have** ‹$z \gtrsim Y$› **by** (*simp add*: *supRs-def*)
from *this* ‹*supRs* b Y X› ‹$z \in X$› *supRs-def*[*of* b Y X]
have ‹$z \geq b$› **by** *simp*
from ‹*supRs* z $(Y \cup \{x\})$ X › **have** ‹$z \geq x$› **by** (*simp add*: *supRs-def*)
from ‹$z \geq x$› ‹$z \geq b$› **have** ‹$z \gtrsim \{x, b\}$› **by** *auto*
then show ‹*consi* $\{b, x\}$ X› **using** *consi-def*[*of* $\{b,x\}$ X]
using ‹$z \in X$› **by** *blast*
qed

from ‹$b \in X$› *this* *IntrPs-def*[*of* X] **have** ‹$b \in$ *IntrPs* X› **by** *simp*
from ‹*supRs* b Y X› ‹$b \in$ *IntrPs* X› **have** ‹*supRs* b Y (*IntrPs* X)›
by (*simp add*: *supRs-def* *IntrPs-def*)

show ‹($\exists$ $b \in$ (*IntrPs* X). *supRs* b Y (*IntrPs* X))› **proof**
show ‹$b \in$ *IntrPs* X› **using** ‹$b \in$ *IntrPs* X› **by** *auto*
next
show ‹*supRs* b Y (*IntrPs* X)› **using** ‹*supRs* b Y (*IntrPs* X)› **by** *auto*
qed
qed
qed
from *this* *complete-lattice-iff-sup* **show** ‹*complete-latticeR* (*IntrPs* X)› **by** *auto*
qed

lemma *intrinsic-largest*:
$\llbracket$ *ccpo* (X:: ('*a*::*order*) *set*) $\rrbracket \Longrightarrow \exists$ $i \in$ (*IntrPs* X). $i \gtrsim$ (*IntrPs* X)
proof−
fix X :: '*a set* **assume** *Hccpo*: ‹*ccpo* X›
then have *complete-latticeR* (*IntrPs* X) **using** *intrinsic-lattice* **by** *auto*
then have $\exists$ $i \in$ (*IntrPs* X). *infRs* i $\{\}$ (*IntrPs* X) **by** (*simp add*: *complete-latticeR-def*)
then obtain i **where** ‹$i \in$ *IntrPs* X› **and** ‹*infRs* i $\{\}$ (*IntrPs* X)› **by** *auto*
show $\exists$ $i \in$ (*IntrPs* X). $i \gtrsim$ (*IntrPs* X) **proof**

```
  from ‹infRs i {} (IntrPs X)› show ∀ y∈IntrPs X. y ≤ i by(simp add:infRs-def)
   from ‹i ∈ IntrPs X› show ‹i ∈ IntrPs X› by auto
 qed
qed

definition MaxiPs ::‹('a ::order) set ⇒ 'a set›
where MaxiPs X = {x∈ X. ∀ y ∈ X. x ≤ y ⟶ x = y}

lemma greatest-intrinsic-is-sup-maxH1:
⟦ ccpo (X :: ('a ::order) set); i ∈ (IntrPs X); i ≳ (IntrPs X) ⟧
  ⟹ (i ∈ X ∧ infRs i (MaxiPs X) X) proof−
 fix X :: ‹('a ::order) set› fix i
 assume ‹ccpo X› ‹i ∈ (IntrPs X)› ‹i ≳ (IntrPs X)›
 show (i ∈ X ∧ infRs i (MaxiPs X) X) proof
   show ‹i ∈ X› using ‹i ∈ (IntrPs X)› IntrPs-def by blast
   have ‹i ≲ (MaxiPs X)› proof
    fix m assume ‹m ∈ MaxiPs X› then have ‹m ∈ X› by(simp add:MaxiPs-def)
     from this ‹i ∈ (IntrPs X)› have ‹consi {i, m} X› by(simp add: IntrPs-def)
     from ‹i ∈ X› ‹m ∈ X› this ‹ccpo X› have ‹∃ b∈X. supRs b {i,m} X›
       by (simp add:ccpo-def)
     then obtain b where ‹b ∈ X› and ‹supRs b {i,m} X› by auto
     from this have ‹i ≤ b› by(simp add: supRs-def)
     from ‹supRs b {i,m} X› have ‹m ≤ b› by(simp add: supRs-def)
     from this ‹m ∈ MaxiPs X› ‹b ∈ X› have m = b by (simp add: MaxiPs-def)

     from this ‹i ≤ b› show ‹i ≤ m› by auto
   qed

   from max-elem-in-ccpo[of X i] ‹ccpo X› ‹i ∈ X› MaxiPs-def[of X]
      have (MaxiPs X) ≠ {} by auto
   from this ‹ccpo X› ccpo-nonempty-has-inf MaxiPs-def
   have ∃ j ∈ X. infRs j (MaxiPs X) X by auto
   then obtain j where ‹j ∈ X› and ‹infRs j (MaxiPs X) X› by auto
   from ‹infRs j (MaxiPs X) X› ‹i ≲ (MaxiPs X)› ‹i ∈ X›
   have ‹i ≤ j› by (simp add: infRs-def)

   have ∀ x ∈ X. consi {j, x} X proof
     fix x assume ‹x ∈ X›
     from max-elem-in-ccpo[of X x] ‹ccpo X› ‹x ∈ X› have
        ∃ m∈X. (∀ a∈X. m ≤ a ⟶ m = a) ∧ x ≤ m by auto
     from this obtain m where ‹m ∈ X› and ‹(∀ a∈X. m ≤ a ⟶ m = a)› and
‹x ≤ m› by auto
     from ‹(∀ a∈X. m ≤ a ⟶ m = a)› ‹m ∈ X› have ‹m ∈ MaxiPs X› by(simp
add: MaxiPs-def)
     from ‹infRs j (MaxiPs X) X›  ‹m ∈ MaxiPs X› have ‹j ≤ m› by(simp add:
infRs-def)
      from ‹j ≤ m› ‹x ≤ m› ‹m ∈ X› show consi {j, x} X using consi-def by
blast
   qed
```

from *this IntrPs-def*[*of X*] ‹$j \in X$› **have** ‹$j \in (IntrPs\ X)$› **by** *simp*
from *this* ‹$i \gtrsim (IntrPs\ X)$› **have** ‹$j \leq i$› **by**(*simp add*: *IntrPs-def*)
from ‹$i \leq j$› ‹$j \leq i$› **have** ‹$i = j$› **by** *auto*
from *this* ‹*infRs j* (*MaxiPs X*) *X*› **show** *infRs i* (*MaxiPs X*) *X* **by** *auto*
qed
qed

3 Preliminary Matters (continued)

datatype (*'function-type*, *'constant-type*) *tm*
= *Var nat*
| *Const 'constant-type*
| *Fun 'function-type* ‹(*'function-type*, *'constant-type*) *tm list*›

datatype (*'function-type*, *'constant-type*, *'relation-type*) *fm*
= *Rel 'relation-type* ‹(*'function-type*, *'constant-type*) *tm list*›
| *Equ* ‹(*'function-type*, *'constant-type*) *tm*› ‹(*'function-type*, *'constant-type*) *tm*›
| *Fal*
| *And* ‹(*'function-type*, *'constant-type*, *'relation-type*) *fm*› ‹(*'function-type*, *'constant-type*, *'relation-type*) *fm*›
| *Neg* ‹(*'function-type*, *'constant-type*, *'relation-type*) *fm*›
| *Forall nat* ‹(*'function-type*, *'constant-type*, *'relation-type*) *fm*›

fun *freevar-tm* :: (*'a*, *'b*) *tm* ⇒ *nat set* **where**
freevar-tm (*Var n*) = { *n* } |
freevar-tm (*Const b*) = {} |
freevar-tm (*Fun f-symb term-list*) = ⋃ (*set* (*map freevar-tm term-list*))

fun *freevar* :: (*'a*, *'b*, *'c*) *fm* ⇒ *nat set* **where**
freevar (*Rel r-symb term-list*) = ⋃ (*set* (*map freevar-tm term-list*)) |
freevar (*Equ tm1 tm2*) = (*freevar-tm tm1*) ∪ (*freevar-tm tm2*) |
freevar (*And fm1 fm2*) = (*freevar fm1*) ∪ (*freevar fm2*) |
freevar (*Neg f*) = (*freevar f*) |
freevar Fal = {} |
freevar (*Forall m f*) = (*freevar f*) − {*m*}

fun *contvar* :: (*'a*, *'b*, *'c*) *fm* ⇒ *nat set* **where**
contvar (*Rel r-symb term-list*) = ⋃ (*set* (*map freevar-tm term-list*)) |
contvar (*Equ tm1 tm2*) = (*freevar-tm tm1*) ∪ (*freevar-tm tm2*) |
contvar (*And fm1 fm2*) = (*contvar fm1*) ∪ (*contvar fm2*) |
contvar (*Neg f*) = (*contvar f*) |
contvar Fal = {} |
contvar (*Forall m f*) = (*contvar f*) ∪ {*m*}

fun *freevar-tmL* :: (*'a*, *'b*) *tm* ⇒ *nat list* **where**
freevar-tmL (*Var n*) = [*n*] |
freevar-tmL (*Const b*) = [] |
freevar-tmL (*Fun f-symb term-list*) = *remdups* (*concat* (*map freevar-tmL term-list*)

)

lemma *freevar-tm-id*: ‹*set* (*freevar-tmL t*) = *freevar-tm t*›
  by(*induction t*; *simp*)

fun *freevarL* :: ('*a*, '*b*, '*c*) *fm* ⇒ *nat list* **where**
freevarL (*Rel r-symb term-list*) = *remdups* (*concat* (*map freevar-tmL term-list*)) |
freevarL (*Equ tm1 tm2*) = *remdups* (*concat*[(*freevar-tmL tm1*), (*freevar-tmL tm2*)]) |
freevarL (*And fm1 fm2*) = *remdups* (*concat*[(*freevarL fm1*), (*freevarL fm2*)]) |
freevarL (*Neg f*) = (*freevarL f*) |
freevarL Fal = [] |
freevarL (*Forall m f*) = *removeAll m* (*freevarL f*)

lemma *freevar-id*: ‹*set* (*freevarL f*) = *freevar f*›
  by(*induction f*; *simp add*: *freevar-tm-id*)

lemma *freevar-contvar*: *freevar f* ⊆ *contvar f*
  by(*induction f*; *simp*; *auto*)

definition *sentence* **where**
sentence f1 ≡ (*freevar f1* = {})

type-synonym '*v assignment* = *nat* ⇒ '*v*
type-synonym ('*v*, '*a*,'*b*,'*c*) *const-mod* = '*b* ⇒ '*v*
type-synonym ('*v*, '*a*,'*b*,'*c*) *func-mod* = '*a* ⇒ ('*v list*) ⇒ '*v*
type-synonym ('*v*, '*a*,'*b*,'*c*) *rela-mod-tau* = '*c* ⇒ ('*v list*) ⇒ *bool*

The idea here is that the value of e.g. func_mod Fsymb should be set to undefined in a case where the lenght of the argument (which is a list) does not correspond to the arity of Fsymb. Also, if some object of type 'v comes not from the domain set and is in the argument list, we expect undefined as image

Rela_mod_tau is just how one would do it using the build-in bool type of isabelle. However, this is just a deadend example because there True ≤ False, which is undesired in our context.

'v is the type of the domain D ; 'b is the type of constant symbols; 'a is the type of function symbols; 'c is the type of relation symbols

type-synonym ('*v*, '*a*,'*b*,'*c*) *mod1*
  = ‹('*v*, '*a*,'*b*,'*c*) *const-mod* × ('*v*, '*a*,'*b*,'*c*) *func-mod* × ('*v*, '*a*,'*b*,'*c*) *rela-mod-tau*›

fun *value-tm* :: ‹'*v assignment* ⇒ (('*v*, '*a*,'*b*,'*c*) *const-mod* × ('*v*, '*a*,'*b*,'*c*) *func-mod*) ⇒ ('*a*, '*b*) *tm* ⇒ '*v*› **where**
(*value-tm s* (*Cw*, *Fw*) (*Var n*)) = (*s n*) |
(*value-tm s* (*Cw*, *Fw*) (*Const c*)) = (*Cw c*) |
(*value-tm s* (*Cw*, *Fw*) (*Fun f-symb term-list*))
    = (*Fw f-symb* (*map* (λ *x*. *value-tm s* (*Cw*, *Fw*) *x*) *term-list*))

lemma *value-tm-locdet*: ‹⟦ ∀ m ∈ freevar-tm t. s1 m = s2 m ⟧
⟹ value-tm s1 (Cw, Fw) t = value-tm s2 (Cw, Fw) t›
apply(*induction t*; *simp*) **by** (*metis* (*mono-tags*, *lifting*) *map-eq-conv*)

type-synonym ('v, 'a,'b,'c, 'mybool) *rela-mod* =
'c ⇒ ('v list) ⇒ 'mybool

type-synonym ('v, 'a,'b,'c, 'mybool) *model*
= ‹'v set × ('v, 'a,'b,'c) const-mod × ('v, 'a,'b,'c) func-mod × ('v, 'a,'b,'c, 'mybool) rela-mod›

type-synonym ('v, 'mybool) *scheme*
= ‹'mybool × 'mybool × ('mybool ⇒ 'mybool) × ('mybool ⇒ 'mybool ⇒ 'mybool) × ('v set ⇒ ('v ⇒ 'mybool) ⇒ 'mybool)›

fun *value-fm* :: ‹('v, 'mybool) scheme ⇒ 'v assignment ⇒ ('v, 'a, 'b, 'c, 'mybool) model ⇒ ('a, 'b, 'c) fm ⇒ 'mybool› **where**
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*, *Fw*, *Rw*) *Fal* = *myFalse* |
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*, *Fw*, *Rw*) (*Rel r-symb term-list*) = (*Rw r-symb* (*map* (λ *x*. *value-tm w* (*Cw*,*Fw*) *x*) *term-list*)) |
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*, *Fw*, *Rw*) (*Equ tm1 tm2*) = (*if* (*value-tm w* (*Cw*,*Fw*) *tm1* = *value-tm w* (*Cw*,*Fw*) *tm2*) *then myTrue else myFalse*) |
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*, *Fw*, *Rw*) (*And fm1 fm2*) = (*myAnd* (*value-fm* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*,*Fw*,*Rw*) *fm1*) (*value-fm* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*,*Fw*,*Rw*) *fm2*)) |
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*, *Fw*, *Rw*) (*Neg f*) = (*myNot* (*value-fm* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*,*Fw*,*Rw*) *f*)) |
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*, *Fw*, *Rw*) (*Forall m f*) = (*myUni D* (λ *v*. *value-fm* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) (λ *k*. *if k*=*m then v else w k*) (*D*,*Cw*,*Fw*,*Rw*) *f*))

fun *τ-not* ::‹bool2 ⇒ bool2› **where**
τ-not t2 = *f2* |*τ-not f2* = *t2*

lemma *τ-not-monot*: ‹b1 ≤ b2 ⟹ τ-not b1 ≤ τ-not b2›
by(*cases b1*; *cases b2*; *simp add*: *less-eq-bool2-def*)

fun *τ-and* ::‹bool2 ⇒ bool2 ⇒ bool2› **where**
‹ τ-and t2 t2 = t2› | ‹ τ-and - - = f2›

lemma *τ-and-monot*: ‹⟦ a1 ≤ a2; b1 ≤ b2 ⟧ ⟹ τ-and a1 b1 ≤ τ-and a2 b2›

by(*cases a1*; *cases a2*; *cases b1*; *cases b2*; *simp add*: *less-eq-bool2-def*)

fun τ*-forall* ::⟨(′*v set*) ⇒ (′*v* ⇒ *bool2*) ⇒ *bool2*⟩ **where**
⟨(τ*-forall D f*) = (*if* (∀ *v* ∈ *D*. *f v* = *t2*) *then t2 else f2*) ⟩

lemma ⟨ (λ *x*. *n3*) ≤ (λ *x*. *t3*) ⟩
  by (*simp add*: *le-funI less-eq-bool3-def*)

lemma ⟨ ¬ (λ *x*. *t2*) ≤ (λ *x*. *f2*)⟩
  by (*meson le-funD leq2*.*simps*(*4*) *less-eq-bool2-def*)

lemma τ*-forall-monot*:
  ⟨(*fb1* :: (′*v* ⇒ *bool2*)) ≤ (*fb2* :: (′*v* ⇒ *bool2*))
      ⟹ (τ*-forall D fb1*) ≤ (τ*-forall D fb2*)⟩
   apply(*cases* τ*-forall D fb1*; *cases* τ*-forall D fb2*; *simp add*: *less-eq-bool2-def le-fun-def*)
  apply(*smt bool2*.*exhaust image-cong leq2*.*simps*(*4*) *range-eq-singletonD*)
  by (*smt bool2*.*exhaust image-cong leq2*.*simps*(*3*) *range-eq-singletonD*)

abbreviation τ :: ⟨(′*v*, *bool2*) *scheme*⟩
  where ⟨τ ≡ (*f2*, *t2*, τ*-not*, τ*-and*, τ*-forall*)⟩

fun μ*-not*::⟨*bool3* ⇒ *bool3*⟩ **where**
⟨μ*-not t3* = *f3*⟩ |
⟨μ*-not f3* = *t3*⟩ |
⟨μ*-not n3* = *n3*⟩

lemma μ*-not-monot*: ⟨*b1* ≤ *b2* ⟹ μ*-not b1* ≤ μ*-not b2*⟩
   by (*smt* μ*-not*.*elims leq3*.*simps*(*2*) *leq3*.*simps*(*3*) *leq3*.*simps*(*4*) *leq3*.*simps*(*5*) *leq3*.*simps*(*6*) *leq3*.*simps*(*7*) *leq3*.*simps*(*8*) *leq3*.*simps*(*9*) *less-eq-bool3-def*)

fun μ*-and*::⟨*bool3* ⇒ *bool3* ⇒ *bool3*⟩ **where**
⟨μ*-and n3* - = *n3*⟩ | ⟨μ*-and* - *n3* = *n3*⟩ |
⟨μ*-and t3 t3* = *t3*⟩ | ⟨μ*-and t3 f3* = *f3*⟩ |
⟨μ*-and f3 t3* = *f3*⟩ | ⟨μ*-and f3 f3* = *f3*⟩

lemma μ*-and-monot*: ⟨⟦ *fm1* ≤ *fm2* ; *g1* ≤ *g2* ⟧ ⟹ μ*-and fm1 g1* ≤ μ*-and fm2 g2*⟩
  by(*cases fm1*; *cases fm2*; *cases g1*; *cases g2*; *simp add*:*less-eq-bool3-def*)

fun μ*-forall* :: ⟨(′*v set*) ⇒ (′*v* ⇒ *bool3*) ⇒ *bool3*⟩ **where**
μ*-forall D f* = (*if* (∀ *v* ∈ *D*. *f v*= *t3*) *then t3 else*
       (*if* (∃ *v* ∈ *D*. *f v*= *n3*) *then n3 else f3*))

lemma μ*-forall-monot*:⟨*f1* ≤ *fm2* ⟹ μ*-forall D f1* ≤ μ*-forall D fm2*⟩
  by(*cases* μ*-forall D f1*; *cases* μ*-forall D fm2*; *simp add*: *less-eq-bool3-def*;

metis (*full-types*) *bool3.exhaust le-fun-def leq3.simps less-eq-bool3-def*)

lemma ‹*range f* = {*t3*} ⟹ *μ-forall D f* = *t3*› **by** *auto*
lemma ‹*f* ‘ *D* = {*f3*} ⟹ *μ-forall D f* = *f3*›
by (*metis μ-forall.simps bex-imageD insert-image leq3.simps*(*3*) *singletonI singleton-insert-inj-eq*′)
lemma ‹*f* ‘ *D* = {*n3*} ⟹ *μ-forall D f* = *n3*›
by (*metis μ-forall.simps bex-imageD insert-image leq3.simps*(*1*) *singletonI singleton-insert-inj-eq*′)
lemma ‹*f* ‘ *D* = {*n3*, *t3*} ⟹ *μ-forall D f* = *n3*›
using *image-iff* [*of n3*] **by** (*metis μ-forall.simps insertI1*)

lemma ‹ *f* ‘ *D* = {*n3*, *f3*} ⟹ *μ-forall D f* = *n3*›
using *image-iff* [*of n3*] **by** (*metis μ-forall.simps insertI1*)

lemma ‹*f* ‘ *D* = {*f3*, *t3*} ⟹ *μ-forall D f* = *f3*› **proof**−
assume *A*: *f* ‘ *D* = {*f3*, *t3*}
from *A* **have** ‹∃ *v1* ∈ *D*. *f v1* = *f3*› **using** *image-iff* **by** *fastforce*
then obtain *v1* **where** *1*: ‹*v1* ∈ *D* ∧ *f v1* = *f3*› **by** *auto*
from *A* **have** ‹∃ *v2* ∈ *D*. *f v2* = *t3*› **using** *image-iff* **by** *fastforce*
then obtain *v2* **where** *2*: ‹*v2* ∈ *D* ∧ *f v2* = *t3*› **by** *auto*

from *1* **have** *3*: ¬ (∀ *v* ∈ *D*. *f v*= *t3*) **by** *fastforce*
from *A* **have** *4*: ¬ (∃ *v* ∈ *D*. *f v*= *n3*)
by (*metis* (*mono-tags*, *lifting*) *bool3.distinct*(*1*) *bool3.simps*(*4*) *insert-absorb insert-iff insert-image singleton-insert-inj-eq*′)

show *μ-forall D f* = *f3* **using** *3 4* **by** *simp*
qed
lemma ‹*f* ‘ *D* = {*n3*, *t3*, *f3*} ⟹ *μ-forall D f* = *n3*›
using *image-iff* [*of n3*] **by** (*metis μ-forall.simps insertI1*)

abbreviation *μ*:: ‹(′*v*, *bool3*) *scheme*›
where ‹*μ* ≡ (*f3*, *t3*, *μ-not*, *μ-and*, *μ-forall*)›

fun *κ-not*::‹*bool3* ⇒ *bool3*› **where**
‹*κ-not t3* = *f3*› |
‹*κ-not f3* = *t3*› |
‹*κ-not n3* = *n3*›

lemma *κ-not-monot*: ‹*b1* ≤ *b2* ⟹ *κ-not b1* ≤ *κ-not b2*›
by (*smt κ-not.elims leq3.simps*(*2*) *leq3.simps*(*3*) *leq3.simps*(*4*) *leq3.simps*(*5*) *leq3.simps*(*6*) *leq3.simps*(*7*) *leq3.simps*(*8*) *leq3.simps*(*9*) *less-eq-bool3-def*)

fun *κ-and*::‹*bool3* ⇒ *bool3* ⇒ *bool3*› **where**
‹*κ-and f3* - = *f3*› | ‹*κ-and* - *f3* = *f3*› |
‹*κ-and t3 t3* = *t3*› | ‹*κ-and t3 n3* = *n3*› |
‹*κ-and n3 t3* = *n3*› | ‹*κ-and n3 n3* = *n3*›

lemma *κ-and-monot*: ‹⟦ *f1* ≤ *fm2* ; *g1* ≤ *g2* ⟧ ⟹ *κ-and f1 g1* ≤ *κ-and fm2 g2*›
by(*cases f1*; *cases fm2*; *cases g1*; *cases g2*; *simp add*:*less-eq-bool3-def*)

fun *κ-forall* :: ‹(′*v set*) ⇒ (′*v* ⇒ *bool3*) ⇒ *bool3*› **where**
κ-forall D f = (*if* (∀ *v* ∈ *D*. *f v* = *t3*)
    then t3 else (*if* (∃ *v* ∈ *D*. *f v* = *f3*) *then f3 else n3*))

lemma *κ-forall-monot*:‹*f1* ≤ *fm2* ⟹ *κ-forall D f1* ≤ *κ-forall D fm2*›
  by(*cases κ-forall D f1*; *cases κ-forall D fm2*; *simp add*: *less-eq-bool3-def*;
   metis (*full-types*) *bool3.exhaust le-fun-def leq3.simps less-eq-bool3-def*)

lemma ‹*f*‘ *D* = {*t3*} ⟹ *κ-forall D f* = *t3*› **by** *auto*
lemma ‹*f*‘ *D* = {*f3*} ⟹ *κ-forall D f* = *f3*› **using** *image-iff*[*of f3*] **by** *fastforce*
lemma ‹*f*‘ *D* = {*n3*} ⟹ *κ-forall D f* = *n3*› **proof**−
  assume *H*: *f*‘ *D* = {*n3*}
  then have ∃ *d* ∈ *D*. *f d* = *n3* **using** *image-iff*[*of n3*] **by** *fastforce*
  then have *1*: ¬(∀ *v* ∈ *D*. *f v* = *t3*) **by** *fastforce*
  from *H* **have** *2*: ¬ (∃ *v* ∈ *D*. *f v* = *f3*) **by** *force*
  from *1 2* **show** *?thesis* **by** *simp*
qed

lemma ‹*f*‘ *D* = {*n3*, *t3*} ⟹ *κ-forall D f* = *n3*› **proof**−
  assume *A*: *f*‘ *D* = {*n3*,*t3*}
  from *A* **have** ‹∃ *v1* ∈ *D*. *f v1* = *n3*› **using** *image-iff*[*of n3*] **by** *fastforce*
  then obtain *v1* **where** *1*: ‹*v1* ∈ *D* ∧ *f v1* = *n3*› **by** *auto*
  from *A* **have** *2*: ¬(∃ *v*∈ *D*. *f v* = *f3*) **by** *force*
  from *1 2* **show** *?thesis* **by** (*metis κ-forall.simps*)
qed
lemma ‹*f*‘ *D* = {*n3*, *f3*} ⟹ *κ-forall D f* = *f3*›
proof−
  assume *f*‘ *D* = {*n3*, *f3*}
  then have ∃ *d* ∈ *D*. *f d* = *f3* **using** *image-iff*[*of f3*] **by** *fastforce*
  then show *?thesis* **by** *force*
qed
lemma ‹ *f*‘ *D* = {*f3*, *t3*} ⟹ *κ-forall D f* = *f3*›
proof−
  assume *f*‘ *D* = {*f3*, *t3*}
  then have ∃ *d* ∈ *D*. *f d* = *f3* **using** *image-iff*[*of f3*] **by** *fastforce*
  then show *?thesis* **by** *force*
qed

lemma ‹*f*‘ *D* = {*n3*, *t3*, *f3*} ⟹ *κ-forall D f* = *f3*›
proof−
  assume *f*‘ *D* = {*n3*, *t3*, *f3*}
  then have ∃ *d* ∈ *D*. *f d* = *f3* **using** *image-iff*[*of f3*] **by** *fastforce*
  then show *?thesis* **by** *force*
qed

abbreviation *κ*:: ‹(′*v*, *bool3*) *scheme*›

where ‹κ ≡ (*f3*, *t3*, *κ-not*, *κ-and*, *κ-forall*)›

fun *ν-not*::‹*bool4* ⇒ *bool4*› **where**
ν-not t4 = *f4* | *ν-not f4* = *t4* | *ν-not n4* = *n4* | *ν-not b4* = *b4*

fun *ν-and*::‹*bool4* ⇒ *bool4* ⇒ *bool4*› **where**
ν-and t4 (*b*:: *bool4*) = *b* | *ν-and* (*b*:: *bool4*) *t4* = *b* |
ν-and f4 (*b*:: *bool4*) = *f4* | *ν-and* (*b*:: *bool4*) *f4* = *f4* |
ν-and n4 n4 = *n4* | *ν-and b4 b4* = *b4* | *ν-and b4 n4* = *f4*|
ν-and n4 b4 = *f4*

lemma *ν-and-monot*: ‹⟦ *f1* ≤ *fm2* ; *g1* ≤ *g2* ⟧ ⟹ *ν-and f1 g1* ≤ *ν-and fm2 g2*›
by(*cases f1*; *cases fm2*; *cases g1*; *cases g2*; *simp-all add*:*less-eq-bool4-def*)

fun *ν-forall* :: ‹('*v set*) ⇒ ('*v* ⇒ *bool4*) ⇒ *bool4*› **where**
ν-forall D f = (*if* (∀ *v* ∈ *D*. *f v* = *t4*) *then t4*
else (*if* ((∃ *v* ∈ *D*. *f v* = *f4*) ∨ ((∃ *v1* ∈ *D*. *f v1* = *n4*) ∧ (∃ *v2* ∈ *D*. *f v2* = *b4*))) *then f4*
else(*if* (∀ *v* ∈ *D*. *f v* = *t4* ∨ *f v* = *n4*) *then n4 else b4*
)))

fun *ν-forall2* :: ‹('*v set*) ⇒ ('*v* ⇒*bool4*)⇒ *bool4*› **where**
ν-forall2 D f = (*if* (*f*‘ *D*) = {*t4*}
then t4 else (*if* (*f4* ∈ (*f*‘ *D*) ∨ {*n4*,*b4*} ⊆ (*f*‘ *D*)) *then f4 else*(
if ((*f*‘ *D*) = {*n4*} ∨ (*f*‘ *D*) = {*t4*, *n4*}) *then n4 else b4*
)))

lemma *ν-forallν-forall2*: ‹*D* ≠ {} ⟹ *ν-forall D f*= *ν-forall2 D f*›
proof−
fix *D* :: ‹'*v set*›
fix *f* :: ‹'*v* ⇒ *bool4*›
assume *H*: ‹*D* ≠ {}›
show ‹*ν-forall D f*= *ν-forall2 D f*› **proof**(*cases* ∀ *v*∈*D*. *f v*= *t4*)
case *True*
from *True* **have** *1*: *ν-forall D f* = *t4* **by** *simp*
from *True H* **have** *f*‘ *D* = {*t4*} **by** *force*
then have *2*: ‹*ν-forall2 D f* = *t4*› **by** *auto*
from *1 2* **show** *?thesis* **by** *auto*
next
case *False*
then have *F1*: ¬ (∀ *v* ∈ *D*. *f v* = *t4*) **by** *auto*
show *?thesis* **proof**(*cases* ((∃ *v* ∈ *D*. *f v* = *f4*) ∨ ((∃ *v1* ∈ *D*. *f v1* = *n4*) ∧ (∃ *v2* ∈ *D*. *f v2* = *b4*))))
case *True*
from *True F1* **have** *1*: *ν-forall D f* = *f4* **by** *simp*
from *True H* **have** (*f4* ∈ (*f*‘ *D*) ∨ {*n4*,*b4*} ⊆ (*f*‘*D*)) **by** *force*
then have *2*: ‹*ν-forall2 D f* = *f4*› **by** *auto*

```
    from 1 2 show ?thesis by auto
  next
    case False
    then have F2: ¬ ((∃ v ∈ D. f v = f4) ∨ ((∃ v1 ∈ D. f v1 = n4) ∧ (∃ v2 ∈
D. f v2 = b4))) by auto
    then show ?thesis proof(cases (∀ v∈D. f v = t4 ∨ f v = n4))
      case True
      from True F1 F2 have 1: ν-forall D f = n4 by simp
      from True F1 have ‹(∀ v ∈ D. f v = n4) ∨ (∃ v1 ∈D . ∃ v2 ∈ D. f v1 =
t4 ∧ f v2 = n4)› by blast
      then have ( ( f ‘D) = {n4} ∨ (f ‘D) = {t4, n4} ) proof
        assume (∀ v∈D. f v = n4) then have (f‘D) = {n4} using H by force
        then show ?thesis by auto
      next
        assume A: (∃ v1 ∈D. ∃ v2∈D. f v1 = t4 ∧ f v2 = n4)
        from this have ‹( f‘D) ⊆ {t4, n4}› using True by blast
        from True have ‹( f‘D) ⊇ {t4, n4}›
          by (metis A empty-subsetI image-eqI insert-subset)
        from ‹( f‘D) ⊆ {t4, n4}› ‹( f‘D) ⊇ {t4, n4}› show ?thesis by auto
      qed
    then have 2: ‹ν-forall2 D f = n4› by auto
    from 1 2 show ?thesis by auto
next
  case False
  from False F1 F2 have 1: ν-forall D f = b4 by simp
  have 2: ν-forall2 D f = b4
   by (smt F2 False ν-forall2.simps imageE insertE insert-absorb insert-not-empty
insert-subset image-eqI)
  from 1 2 show ?thesis by auto
qed
qed
qed
qed

fun leqL4 :: ‹bool4 ⇒ bool4 ⇒ bool› where
‹leqL4 - t4 = True› |
‹leqL4 t4 b4 = False› |
‹leqL4 t4 n4 = False› |
‹leqL4 t4 f4 = False› |
‹leqL4 f4 - = True› |
‹leqL4 n4 f4 = False› |
‹leqL4 b4 f4 = False› |
‹leqL4 n4 b4 = False› |
‹leqL4 b4 n4 = False› |
‹leqL4 b4 b4 = True› |
‹leqL4 n4 n4 = True›

lemma leqL4-refl: leqL4 b b = True by(cases b) auto
lemma leqL4-antisym: ⟦ leqL4 u v ; leqL4 v u ⟧ ⟹ u = v by(cases u; cases v)
```

auto
lemma *leqL4-trans*: ⟦ *leqL4 u v*; *leqL4 v w* ⟧ $\Longrightarrow$ *leqL4 u w* **by**(*cases u*; *cases v*; *cases w*) *auto*

fun *leqL4inf* :: ‹*bool4* ⇒ *bool4* ⇒ *bool4*› **where**
‹*leqL4inf - f4* = *f4*› |
‹*leqL4inf f4 -* = *f4*› |
‹*leqL4inf t4* (*b*:: *bool4*) = *b*› |
‹*leqL4inf* (*b*:: *bool4*) *t4* = *b*› |
‹*leqL4inf n4 b4* = *f4*› |
‹*leqL4inf b4 n4* = *f4*› |
‹*leqL4inf b4 b4* = *b4*› |
‹*leqL4inf n4 n4* = *n4*›

Fold with auto works better with lists than with FiniteSet.fold

fun *InfL4*:: ‹*bool4 list* ⇒ *bool4*› **where**
InfL4 (*X* :: *bool4 list*) = *fold leqL4inf X t4*

lemma ‹ ν*-and b1 b2* = *InfL4* [*b1*, *b2*] ›
by(*cases b1*; *cases b2*; *auto*)

lemma *H1*: ‹*f* ‘*D* = { *b* :: *bool4*} $\Longrightarrow$ ν*-forall2 D f* = *InfL4* [*b*]› **by**(*cases b*; *auto*)
lemma *H2*: ‹*f* ‘*D* = { *b1* :: *bool4*, *b2* :: *bool4* }
$\Longrightarrow$ ν*-forall2 D f* = *InfL4* [*b1*, *b2*]› **by**(*cases b1*; *cases b2*; *auto*)
lemma *H3*: ‹*f* ‘*D* = { *b1* :: *bool4*, *b2* :: *bool4*, *b3*:: *bool4* }
$\Longrightarrow$ ν*-forall2 D f* = *InfL4* [*b1*, *b2*, *b3*]›
by(*cases b1*; *cases b2*; *cases b3*; *auto*)
lemma *H4*: ‹*f* ‘*D* = {*n4*, *b4*, *t4*, *f4*} $\Longrightarrow$ ν*-forall2 D f* = *InfL4* [*n4*,*b4*,*t4*,*f4*]› **by** *auto*

lemma ‹*D* ≠ {} $\Longrightarrow$ *f* ‘*D* = { *b* :: *bool4*} $\Longrightarrow$ ν*-forall D f* = *InfL4* [*b*]›
using *H1* ν*-forall*ν*-forall2* **by** *metis*
lemma ‹*D* ≠ {} $\Longrightarrow$ *f*‘ *D* = { *b1* :: *bool4*, *b2* :: *bool4* }
$\Longrightarrow$ ν*-forall D f* = *InfL4* [*b1*, *b2*]› **using** *H2* ν*-forall*ν*-forall2* **by** *metis*
lemma ‹*D* ≠ {} $\Longrightarrow$ *f*‘*D* = { *b1* :: *bool4*, *b2* :: *bool4*, *b3*:: *bool4* }
$\Longrightarrow$ ν*-forall D f* = *InfL4* [*b1*, *b2*, *b3*]›
using *H3* ν*-forall*ν*-forall2* **by** *metis*
lemma ‹*D* ≠ {} $\Longrightarrow$ *f*‘*D* = {*n4*, *b4*, *t4*, *f4*} $\Longrightarrow$ ν*-forall D f* = *InfL4* [*n4*,*b4*,*t4*,*f4*]›
using *H4* ν*-forall*ν*-forall2* **by** *metis*

lemma ν*-not-monot*: ‹*b1* ≤ *b2* $\Longrightarrow$ ν*-not b1* ≤ ν*-not b2*›
by (*smt* ν*-not.simps bool4.exhaust leq4.elims*(*3*) *leq4.simps less-eq-bool4-def*)

lemma *leq4-mon*: *b1* ≤ *b2* $\Longrightarrow$ *leq4 b1 b2*
by (*simp add*: *less-eq-bool4-def*)

lemma ν*-forall-monot-helper*: ‹ *fm1* ≤ *fm2*
$\Longrightarrow$ *leq4* (ν*-forall D fm1*) (ν*-forall D fm2*)›
proof−

```
  assume ‹fm1 ≤ fm2›
  then have H: ⋀ v. fm1 v ≤ fm2 v using le-funD by metis
  show ‹leq4 (ν-forall D fm1) (ν-forall D fm2)›
  proof(cases ∀ v∈D. fm1 v= t4)
    case True
    then have T1: ∀ v∈D. fm1 v= t4 by simp
    then show ?thesis proof(cases ∀ v∈D. fm2 v= t4)
      case True
      then show ?thesis using T1 by simp
    next
      case False
      then have nT2: ¬ (∀ v∈D. fm2 v= t4) by auto
      then show ?thesis proof(cases ((∃ v ∈ D. fm2 v = f4) ∨ ((∃ v1 ∈ D. fm2
v1 = n4) ∧ (∃ v2 ∈ D. fm2 v2 = b4))))
        case True
        then show ?thesis proof
          assume (∃ v ∈ D. fm2 v = f4)
          then obtain v where v ∈ D and Hv: fm2 v = f4 by auto
          from this H have A: fm1 v = f4 ∨ fm1 v = n4
            by (metis bool4.exhaust leq4.simps(13) leq4.simps(7) leq4-mon)
          from T1 have B: fm1 v = t4 using ‹v ∈ D› by simp
          from A B show ?thesis by simp
        next
          assume (∃ v1∈D. fm2 v1 = n4) ∧ (∃ v2∈D. fm2 v2 = b4)
           then obtain v where v ∈ D and Hv: fm2 v = n4 by auto
          from this H have A: fm1 v = n4
            by (metis T1 leq4.simps(5) leq4-mon)
          from T1 have B: fm1 v = t4 using ‹v ∈ D› by simp
          from A B show ?thesis by simp
        qed
      next
        case False
         then have nF2: ¬ ((∃ v∈D. fm2 v = f4) ∨ (∃ v1∈D. fm2 v1 = n4) ∧
(∃ v2∈D. fm2 v2 = b4)) by auto
        show ?thesis proof(cases (∀ v∈D. fm2 v = t4 ∨ fm2 v = n4))
          case True
          from this nT2 have ∃ v ∈ D. fm2 v = n4 by auto
          then obtain v where ‹v ∈ D› and ‹fm2 v = n4› by auto
          from H this have A: fm1 v = n4 by (metis T1 leq4.simps(5) leq4-mon)
          from T1 ‹v ∈ D› have B:fm1 v = t4 by auto
          from A B show ?thesis by simp
        next
          case False
          then have ν-forall D fm2 = b4 using nF2 nT2 by simp
          then show ?thesis by simp
        qed
      qed
    qed
next
```

case *False*
then have *nT1*: $\neg$ ($\forall v{\in}D.$ *fm1 v* = *t4*) **by** *auto*
show *?thesis* **proof** (*cases* (($\exists$ *v* $\in$ *D*. *fm1 v* = *f4*) $\vee$ (($\exists$ *v1* $\in$ *D*. *fm1 v1* = *n4*) $\wedge$ ($\exists$ *v2* $\in$ *D*. *fm1 v2* = *b4*))))
case *True*
then have *F1*: (($\exists$ *v* $\in$ *D*. *fm1 v* = *f4*) $\vee$ (($\exists$ *v1* $\in$ *D*. *fm1 v1* = *n4*) $\wedge$ ($\exists$ *v2* $\in$ *D*. *fm1 v2* = *b4*))) **by** *auto*
show *?thesis* **proof**(*cases* $\forall$ *v*$\in$*D*. *fm2 v*= *t4*)
case *True*
then show *?thesis* **by** (*metis F1 H leq4.simps*(*12*) *leq4.simps*(*9*) *leq4-mon*)
next
case *False*
then have *nT2*: $\neg$ ($\forall$ *v*$\in$*D*. *fm2 v*= *t4*) **by** *auto*
then show *?thesis* **proof**(*cases* (($\exists$ *v* $\in$ *D*. *fm2 v* = *f4*) $\vee$ (($\exists$ *v1* $\in$ *D*. *fm2 v1* = *n4*) $\wedge$ ($\exists$ *v2* $\in$ *D*. *fm2 v2* = *b4*))))
case *True*
then have *1*: ‹ν*-forall D fm2* = *f4*› **by** *fastforce*
from *F1* **have** *2*: ‹ν*-forall D fm1* = *f4*› **by** *fastforce*
from *1 2* **show** *?thesis* **by** *simp*
next
case *False*
then have *nF2*: $\neg$ (($\exists$ *v*$\in$*D*. *fm2 v* = *f4*) $\vee$ ($\exists$ *v1*$\in$*D*. *fm2 v1* = *n4*) $\wedge$ ($\exists$ *v2*$\in$*D*. *fm2 v2* = *b4*)) **by** *auto*
show *?thesis* **proof**(*cases* ($\forall$ *v*$\in$*D*. *fm2 v* = *t4* $\vee$ *fm2 v* = *n4*))
case *True*
from *this* **have** *A*: ($\forall$ *v*$\in$*D*. *fm1 v* = *t4* $\vee$ *fm1 v* = *n4*) **using** *H*
by (*metis bool4.exhaust leq4.simps*(*11*) *leq4.simps*(*12*) *leq4.simps*(*8*) *leq4.simps*(*9*) *leq4-mon*)
from *F1* **show** *?thesis* **using** *A* **by** *auto*

Contradiction with both branches of Falsity of fm1

next
case *False*
then have ν*-forall D fm2* = *b4* **using** *nF2 nT2* **by** *simp*
then show *?thesis* **by** *simp*
qed
qed
qed
next
case *False*
then have *nF1*: $\neg$ (($\exists$ *v* $\in$ *D*. *fm1 v* = *f4*) $\vee$ (($\exists$ *v1* $\in$ *D*. *fm1 v1* = *n4*) $\wedge$ ($\exists$ *v2* $\in$ *D*. *fm1 v2* = *b4*))) **by** *auto*
then show *?thesis* **proof**(*cases* ($\forall$ *v* $\in$ *D*. *fm1 v* = *t4* $\vee$ *fm1 v* = *n4*))
case *True*
then have ν*-forall D fm1* = *n4* **using** *nF1 nT1* **by** *simp*
then show *?thesis* **by** *simp*
next
case *False*
then have *nN1*: $\neg$ ($\forall$ *v* $\in$ *D*. *fm1 v* = *t4* $\vee$ *fm1 v* = *n4*) **by** *auto*

show *?thesis* **proof**(*cases* $\forall\ v{\in}D.\ fm2\ v{=}\ t4$)
case *True*
from *this H* **have** $\forall\ v\ {\in}D.\ fm1\ v \leq t4$ **by** *metis*
hence $\forall\ v \in D.\ fm1\ v = n4 \lor fm1\ v = t4$
by (*metis bool4.exhaust leq4.simps*(*12*) *leq4-mon nF1*)
then show *?thesis* **using** *nT1 nF1* **by** *auto*

Some contradiction....

next
case *False*
then have *nT2*: $\neg\ (\forall\ v{\in}D.\ fm2\ v{=}\ t4)$ **by** *auto*
then show *?thesis* **proof**(*cases* $((\exists\ v \in D.\ fm2\ v = f4) \lor ((\exists\ v1 \in D.\ fm2\ v1 = n4) \land (\exists\ v2 \in D.\ fm2\ v2 = b4))))$
case *True*
then show *?thesis* **proof**
assume $\exists\, v{\in}D.\ fm2\ v = f4$
then obtain *v* **where** $\langle v \in D\rangle$ **and** $\langle fm2\ v = f4\rangle$ **by** *auto*
from *this H* **have** *A*: $fm1\ v = f4 \lor fm1\ v = n4$
by (*metis bool4.exhaust leq4.simps*(*13*) *leq4.simps*(*7*) *leq4-mon*)
from *nF1* **have** *Res1*: $(\forall\, v{\in}D.\ fm1\ v \neq f4)$ **by** *simp*
from *this A* $\langle v \in D\rangle$ **have** *B*: $fm1\ v = n4$ **by** *auto*
from *nF1* **have** $((\forall\ v1 \in D.\ fm1\ v1 \neq n4) \lor (\forall\ v2 \in D.\ fm1\ v2 \neq b4))$ **by** *simp*
from *this B* $\langle v \in D\rangle$ **have** *Res2*: $(\forall\ v \in D.\ fm1\ v \neq b4)$ **by** *auto*
from *Res1 Res2* **show** *?thesis* **using** *nN1* $\langle v \in D\rangle$
using *bool4.exhaust* **by** *blast*
next
assume *As*: $(\exists\, v1{\in}D.\ fm2\ v1 = n4) \land (\exists\, v2{\in}D.\ fm2\ v2 = b4)$
from *As* **obtain** *v1* **where** $\langle v1 \in D\rangle$ **and** $fm2\ v1 = n4$ **by** *auto*
from *this H* **have** *B*: $fm1\ v1 = n4$
by (*metis bool4.exhaust leq4.simps*(*11*) *leq4.simps*(*5*) *leq4.simps*(*8*) *leq4-mon*)
from *nF1* **have** *Res1*: $(\forall\, v{\in}D.\ fm1\ v \neq f4)$ **by** *simp*
from *nF1* **have** $((\forall\ v1 \in D.\ fm1\ v1 \neq n4) \lor (\forall\ v2 \in D.\ fm1\ v2 \neq b4))$ **by** *simp*
from *this B* $\langle v1 \in D\rangle$ **have** *Res2*: $(\forall\ v \in D.\ fm1\ v \neq b4)$ **by** *auto*
from *Res1 Res2* **show** *?thesis* **using** *nN1* $\langle v1 \in D\rangle$
using *bool4.exhaust* **by** *blast*
qed
next
case *False*
then have *nF2*: $\neg\ ((\exists\, v{\in}D.\ fm2\ v = f4) \lor (\exists\, v1{\in}D.\ fm2\ v1 = n4) \land (\exists\, v2{\in}D.\ fm2\ v2 = b4))$ **by** *auto*
show *?thesis* **proof**(*cases* $(\forall\ v{\in}D.\ fm2\ v = t4 \lor fm2\ v = n4)$)
case *True*
from *this* **have** *A*: $(\forall\ v{\in}D.\ fm1\ v = t4 \lor fm1\ v = n4)$ **using** *H*
by (*metis bool4.exhaust leq4.simps*(*11*) *leq4.simps*(*12*) *leq4.simps*(*8*) *leq4.simps*(*9*) *leq4-mon*)
from *nT1 nN1* **show** *?thesis* **using** *A* **by** *auto*

Contradiction with both branches of Falsity of fm1

```
        next
          case False
          then have ν-forall D fm2 = b4 using nF2 nT2 by simp
          then show ?thesis by simp
        qed
      qed
    qed
   qed
  qed
qed
qed
```

```
lemma ν-forall-monot: ‹f1 ≤ fm2
      ⟹ (ν-forall D f1) ≤ (ν-forall D fm2)›
proof−
  assume ‹f1 ≤ fm2›

  then have leq4 (ν-forall D f1) (ν-forall D fm2) by(metis ν-forall-monot-helper)
  then show (ν-forall D f1) ≤ (ν-forall D fm2) using less-eq-bool4-def by simp
qed
```

```
abbreviation ν:: ‹('v, bool4) scheme›
  where ‹ν ≡ (f4, t4, ν-not, ν-and, ν-forall)›
```

```
fun value-fm-τ :: ‹'v assignment ⇒ ('v, 'a, 'b, 'c, bool2) model ⇒ ('a, 'b, 'c) fm
⇒ bool2› where
‹value-fm-τ w (D, Cw, Fw, Rw) f = value-fm τ w (D, Cw, Fw, Rw) f›
```

```
fun value-fm-κ :: ‹'v assignment ⇒ ('v, 'a, 'b, 'c, bool3) model ⇒ ('a, 'b, 'c) fm
⇒ bool3› where
‹value-fm-κ w (D, Cw, Fw, Rw) f = value-fm κ w (D, Cw, Fw, Rw) f›
```

```
fun value-fm-μ :: ‹'v assignment ⇒ ('v, 'a, 'b, 'c, bool3) model ⇒ ('a, 'b, 'c) fm
⇒ bool3› where
‹value-fm-μ w (D, Cw, Fw, Rw) f = value-fm μ w (D, Cw, Fw, Rw) f›
```

```
fun value-fm-ν :: ‹'v assignment ⇒ ('v, 'a, 'b, 'c, bool4) model ⇒ ('a, 'b, 'c) fm
⇒ bool4› where
‹value-fm-ν w (D, Cw, Fw, Rw) f = value-fm ν w (D, Cw, Fw, Rw) f›
```

```
fun lift23 :: ‹bool2 ⇒ bool3› where
lift23 t2 = t3 |lift23 f2 = f3
```

```
fun lift24 :: ‹bool2 ⇒ bool4› where
lift24 t2 = t4 |lift24 f2 = f4
```

lemma κ*-and-lift-commute*:
‹*lift23* (τ*-and* *A* *B*) = κ*-and* (*lift23* *A*) (*lift23* *B*)›
  by(*cases* *A*; *cases* *B*; *auto*)

lemma κ*-not-lift-commute*:
‹*lift23* (τ*-not* *A*) = κ*-not* (*lift23* *A*)›
  by(*cases* *A*; *auto*)

lemma κ*-forall-lift-commute*:
  ‹*lift23* (τ*-forall* *D* *f*) = κ*-forall* *D* (λ *v*. *lift23* (*f* *v*))›
proof(*cases* ∀ *v*. *f* *v* = *t2*)
  case *True*
  then show *?thesis* **using** *lift23.elims* **by** *auto*
next
  case *False*
  then show *?thesis* **using** *lift23.elims* **by** *auto*
qed

lemma μ*-and-lift-commute*:
‹*lift23* (τ*-and* *A* *B*) = μ*-and* (*lift23* *A*) (*lift23* *B*)›
  by(*cases* *A*; *cases* *B*; *auto*)

lemma μ*-not-lift-commute*:
‹*lift23* (τ*-not* *A*) = μ*-not* (*lift23* *A*)›
  by(*cases* *A*; *auto*)

lemma μ*-forall-lift-commute*:
  ‹*lift23* (τ*-forall* *D* *f*) = μ*-forall* *D* (λ *v*. *lift23* (*f* *v*))›
proof(*cases* ∀ *v*. *f* *v* = *t2*)
  case *True*
  then show *?thesis* **using** *lift23.elims* **by** *simp*
next
  case *False*
  then show *?thesis* **using** *lift23.elims* **by** *auto*
qed

lemma ν*-and-lift-commute*:
‹*lift24* (τ*-and* *A* *B*) = ν*-and* (*lift24* *A*) (*lift24* *B*)›
  by(*cases* *A*; *cases* *B*; *auto*)

lemma ν*-not-lift-commute* :
‹*lift24* (τ*-not* *A*) = ν*-not* (*lift24* *A*)›
  by(*cases* *A*; *auto*)

lemma ν*-forall-lift-commute*:
  ‹*lift24* (τ*-forall* *D* *f*) = ν*-forall* *D* (λ *v*. *lift24* (*f* *v*))›
proof(*cases* ∀ *v*. *f* *v*= *t2*)
  case *True*
  then show *?thesis* **using** *lift24.elims* **by** *auto*

next
  case *False*
  then show *?thesis* **using** *lift24.elims* **by** *auto*
qed

lemma *κ-lift-Rel-commute*:
value-fm-κ w (*D*, *Cw*, *Fw*, *λsymb fmlist*. *lift23* (*Rw symb fmlist*)) *f*
 =*lift23* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) *f*)
proof(*induction f arbitrary*: *w*)
case (*Rel x1 x2*)
  then show *?case* **by** *simp*
next
  case (*Equ x1 x2*)
  then show *?case* **by** *simp*
next
  case *Fal*
  then show *?case* **by** *simp*
next
  case (*And fm1 fm2*)
  then show *?case* **using** *κ-and-lift-commute* **by** *simp*
next
  case (*Neg f*)
  then show *?case* **using** *κ-not-lift-commute* **by** *simp*
next
  case (*Forall m f*)
  then have *IH*: $\bigwedge$ *w*. *value-fm-κ w* (*D*, *Cw*, *Fw*, *λrsymb flist*. *lift23* (*Rw rsymb flist*))
   f = *lift23* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) *f*) **by**(*simp*)
  let *?f* = (*λv*. *value-fm-τ* (*λk*. *if k* = *m then v else w k*) (*D*, *Cw*, *Fw*, *Rw*) *f*)
  have (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) (*Forall m f*)) = *τ-forall D ?f* **by** *auto*
  from *this* **have** *A*: *lift23* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*)
    (*Forall m f*)) = *κ-forall D* (*λ v*. *lift23* (*?f v*)) **using** *κ-forall-lift-commute* **by**(*rule HOL.ssubst*)
  from *this IH* **show** *?case* **by**(*cases* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) (*Forall m f*)); *simp*)
qed

lemma *μ-lift-Rel-commute*:
value-fm-μ w (*D*, *Cw*, *Fw*, *λsymb fmlist*. *lift23* (*Rw symb fmlist*)) *f*
 =*lift23* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) *f*)
proof(*induction f arbitrary*: *w*)
case (*Rel x1 x2*)
  then show *?case* **by** *simp*
next
  case (*Equ x1 x2*)
  then show *?case* **by** *simp*
next
  case *Fal*
  then show *?case* **by** *simp*

next
  case (*And fm1 fm2*)
  then show *?case* **using** *μ-and-lift-commute* **by** *simp*
next
  case (*Neg f*)
  then show *?case* **using** *μ-not-lift-commute* **by** *simp*
next
  case (*Forall m f*)
  then have *IH*: $\bigwedge$ *w. value-fm-μ w* (*D*, *Cw*, *Fw*, *λrsymb flist. lift23* (*Rw rsymb flist*))
    f = *lift23* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) *f*) **by** *auto*
  let *?f* = (*λv. value-fm-τ* (*λk. if k* = *m then v else w k*) (*D*, *Cw*, *Fw*, *Rw*) *f*)
  have (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) (*Forall m f*)) = *τ-forall D ?f* **by** *auto*
  from *this* **have** *A*: *lift23* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*)
      (*Forall m f*)) = *μ-forall D* (*λ v. lift23* (*?f v*)) **using** *μ-forall-lift-commute* **by**(*rule HOL.ssubst*)
  from *this IH* **show** *?case* **by**(*cases* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) (*Forall m f*))) *simp-all*
qed

lemma *ν-lift-Rel-commute*:
value-fm-ν w (*D*, *Cw*, *Fw*, *λsymb fmlist. lift24* (*Rw symb fmlist*)) *f*
  =*lift24* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) *f*)
proof(*induction f arbitrary*: *w*)
case (*Rel x1 x2*)
  then show *?case* **by** *simp*
next
  case (*Equ x1 x2*)
  then show *?case* **by** *simp*
next
  case *Fal*
  then show *?case* **by** *simp*
next
  case (*And fm1 fm2*)
  then show *?case* **using** *ν-and-lift-commute* **by** *simp*
next
  case (*Neg f*)
  then show *?case* **using** *ν-not-lift-commute* **by** *simp*
next
  case (*Forall m f*)
  then have *IH*: $\bigwedge$ *w. value-fm-ν w* (*D*, *Cw*, *Fw*, *λrsymb flist. lift24* (*Rw rsymb flist*))
    f = *lift24* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) *f*) **by** *auto*
  let *?f* = (*λv. value-fm-τ* (*λk. if k* = *m then v else w k*) (*D*, *Cw*, *Fw*, *Rw*) *f*)
  have (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) (*Forall m f*)) = *τ-forall D ?f* **by** *auto*
  from *this* **have** *A*: *lift24* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*)
      (*Forall m f*)) = *ν-forall D* (*λ v. lift24* (*?f v*)) **using** *ν-forall-lift-commute* **by**(*rule HOL.ssubst*)
  from *this IH* **show** *?case* **by**(*cases* (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) (*Forall m*

f))) *simp-all*
qed

lemma *tau-κ-coinc*:
lift23 (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) (*f* :: ('*a*, '*b*, '*c*) *fm*))
 = (*value-fm-κ w* (*D*, *Cw*, *Fw*, (λ *rsymb*. λ *flist*. *lift23* (*Rw rsymb flist*))) *f*)
proof(*induction f*)
case (*Rel x1 x2*)
 then show *?case* **by** *simp*
next
 case (*Equ x1 x2*)
then show *?case* **by** *simp*
next
 case *Fal*
 then show *?case* **by** *simp*
next
 case (*And fm1 fm2*)
 then show *?case* **using** *κ-and-lift-commute* **by** *simp*
next
 case (*Neg f*)
 then show *?case* **using** *κ-not-lift-commute* **by** *simp*
next
 case (*Forall m f*)
 then show *?case* **using** *κ-lift-Rel-commute* **by** *metis*
qed

lemma *tau-μ-coinc*:
lift23 (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) (*f* :: ('*a*, '*b*, '*c*) *fm*))
 = (*value-fm-μ w* (*D*, *Cw*, *Fw*, (λ *rsymb*. λ *flist*. *lift23* (*Rw rsymb flist*))) *f*)
proof(*induction f*)
case (*Rel x1 x2*)
 then show *?case* **by** *simp*
next
 case (*Equ x1 x2*)
then show *?case* **by** *simp*
next
 case *Fal*
 then show *?case* **by** *simp*
next
 case (*And fm1 fm2*)
 then show *?case* **using** *μ-and-lift-commute* **by** *simp*
next
 case (*Neg f*)
 then show *?case* **using** *μ-not-lift-commute* **by** *simp*
next
 case (*Forall m f*)
 then show *?case* **using** *μ-lift-Rel-commute* **by** *metis*
qed

lemma *tau-ν-coinc*:
lift24 (*value-fm-τ w* (*D*, *Cw*, *Fw*, *Rw*) (*f* :: ('*a*, '*b*, '*c*) *fm*))
= (*value-fm-ν w* (*D*, *Cw*, *Fw*, (λ *rsymb*. λ *flist*. *lift24* (*Rw rsymb flist*))) *f*)
proof(*induction f*)
case (*Rel x1 x2*)
then show *?case* **by** *simp*
next
case (*Equ x1 x2*)
then show *?case* **by** *simp*
next
case *Fal*
then show *?case* **by** *simp*
next
case (*And fm1 fm2*)
then show *?case* **using** *ν-and-lift-commute* **by** *simp*
next
case (*Neg f*)
then show *?case* **using** *ν-not-lift-commute* **by** *simp*
next
case (*Forall m f*)
then show *?case* **using** *ν-lift-Rel-commute* **by** *metis*
qed

lemma *monot-μ-κ-leq3*: ($\forall$ *v*. *leq3* ((*fm1* :: '*v* $\Rightarrow$ *bool3*) *v*) ((*fm2* :: '*v* $\Rightarrow$ *bool3*) *v*))
$\Longrightarrow$ *leq3* (*μ-forall D fm1*) (*κ-forall D fm2*)
apply(*cases μ-forall D fm1*) **apply**(*cases κ-forall D fm2*) **apply**(*simp-all*)
apply (*metis* (*full-types*) *bool3.exhaust leq3.simps*(*6*) *leq3.simps*(*7*) *less-eq-bool3-def*)
by(*metis* (*full-types*) *bool3.exhaust leq3.simps*(*2*) *leq3.simps*(*3*) *leq3.simps*(*8*) *leq3.simps*(*9*) *less-eq-bool3-def*)

lemma *monot-μ-κ*: (*fm1* :: '*v* $\Rightarrow$ *bool3*) $\leq$ (*fm2* :: '*v* $\Rightarrow$ *bool3*)
$\Longrightarrow$ (*μ-forall D fm1*) $\leq$ (*κ-forall D fm2*)
proof–
assume *fm1* $\leq$ *fm2*
from *this le-fun-def*[*of fm1 fm2*] **have** ($\forall$ *v*. *fm1 v* $\leq$ *fm2 v*) **by** *simp*
from *this less-eq-bool3-def* **have** $\forall$ *v*. *leq3* (*fm1 v*) (*fm2 v*) **by** *simp*
from *this monot-μ-κ-leq3*[*of fm1 fm2*] **have** *leq3* (*μ-forall D fm1*)
(*κ-forall D fm2*) **by** *simp*
from *this less-eq-bool3-def* **show** (*μ-forall D fm1*) $\leq$ (*κ-forall D fm2*) **by** *simp*
qed

lemma *κ-μ-leq-prop*:
‹($\bigwedge$ *w*. (*value-fm-μ w* (*D*, *Cw*, *Rw*, *Fw*) *f*) $\leq$ (*value-fm-κ w* (*D*, *Cw*, *Rw*, *Fw*) *f*))
$\Longrightarrow$ (*value-fm-μ w* (*D*, *Cw*, *Rw*, *Fw*) (*Forall m f*)) $\leq$ (*value-fm-κ w* (*D*, *Cw*, *Rw*, *Fw*) (*Forall m f*))›
proof–
assume *H*: $\bigwedge$ *w*. (*value-fm-μ w* (*D*, *Cw*, *Rw*, *Fw*) *f*) $\leq$ (*value-fm-κ w* (*D*, *Cw*,

Rw, *Fw*) *f*)
  let *?f2* = λ *v*. *value-fm-*κ (λ *k*. *if k=m then v else w k*) (*D*, *Cw*,*Rw*,*Fw*) *f*
  let *?f1* = λ *v*. *value-fm-*μ (λ *k*. *if k=m then v else w k*) (*D*, *Cw*,*Rw*,*Fw*) *f*
  have *1*: *value-fm-*μ *w* (*D*, *Cw*, *Rw*, *Fw*) (*Forall m f*) = μ*-forall D ?f1* **by** *simp*
  have *2*: *value-fm-*κ *w* (*D*, *Cw*, *Rw*, *Fw*) (*Forall m f*) = κ*-forall D ?f2* **by** *simp*
  from *H* **have** *?f1* $\leq$ *?f2* **by** (*simp add*: *le-funI less-eq-bool3-def*)
 from *this* **have** μ*-forall D ?f1* $\leq$ κ*-forall D ?f2* **using** *less-eq-bool3-def monot-*μ*-*κ
by *blast*
 then show (*value-fm-*μ *w* (*D*, *Cw*, *Rw*, *Fw*) (*Forall m f*)) $\leq$ (*value-fm-*κ *w* (*D*,
Cw, *Rw*, *Fw*) (*Forall m f*))
    using *1 2* **by** *simp*
qed

lemma *val*μ*leq*κ:
(*value-fm-*μ *w* (*D*, *Cw*, *Fw*, *Rw*) *f*) $\leq$ (*value-fm-*κ *w* (*D*, *Cw*, *Fw*, *Rw*) *f*)
proof(*induction f arbitrary*: *w*)
case (*Rel x1 x2*)
  then show *?case*
    by(*cases* (*value-fm-*μ *w* (*D*, *Cw*, *Fw*, *Rw*) (*Rel x1 x2*))) *simp-all*
next
  case (*Equ x1 x2*)
  then show *?case* **by** *simp*
next
  case *Fal*
  then show *?case* **by** *simp*
next
  case (*And fm1 fm2*)
  then have *H1*: $\bigwedge$ *w*. (*value-fm-*μ *w* (*D*, *Cw*, *Fw*, *Rw*) *fm1*) $\leq$ (*value-fm-*κ *w*
(*D*, *Cw*, *Fw*, *Rw*) *fm1*)
   and *H2*: $\bigwedge$ *w*. (*value-fm-*μ *w* (*D*, *Cw*, *Fw*, *Rw*) *fm2*) $\leq$ (*value-fm-*κ *w* (*D*, *Cw*,
Fw, *Rw*) *fm2*) **by** *auto*
  fix *w*
  from *H1*[*of w*] *H2*[*of w*] **show** (*value-fm-*μ *w* (*D*, *Cw*, *Fw*, *Rw*) (*And fm1 fm2*))
$\leq$
        (*value-fm-*κ *w* (*D*, *Cw*, *Fw*, *Rw*) (*And fm1 fm2*))
    by(*cases value-fm-*μ *w* (*D*, *Cw*, *Fw*, *Rw*) *fm1*;
   cases value-fm-μ *w* (*D*, *Cw*, *Fw*, *Rw*) *fm2*;
   cases value-fm-κ *w* (*D*, *Cw*, *Fw*, *Rw*) *fm1*;
   cases value-fm-κ *w* (*D*, *Cw*, *Fw*, *Rw*) *fm2*) (*simp-all add*: *less-eq-bool3-def*)
next
  case (*Neg f*)
  then have *H*: $\bigwedge$ *w*. (*value-fm-*μ *w* (*D*, *Cw*, *Fw*, *Rw*) *f*) $\leq$
            (*value-fm-*κ *w* (*D*, *Cw*, *Fw*, *Rw*) *f*) **by** *auto*
  fix *w*
  from *H*[*of w*] **show** (*value-fm-*μ *w* (*D*, *Cw*, *Fw*, *Rw*) (*Neg f*)) $\leq$
      (*value-fm-*κ *w* (*D*, *Cw*, *Fw*, *Rw*) (*Neg f*))
    by(*cases* (*value-fm-*μ *w* (*D*, *Cw*, *Fw*, *Rw*) *f*);
      cases (*value-fm-*κ *w* (*D*, *Cw*, *Fw*, *Rw*) *f*))
      (*simp-all add*: *less-eq-bool3-def*)

next
  case (*Forall m f*)
  then show *?case* **by**(*rule κ-μ-leq-prop*)
qed

fun *leqMod* ::‹('*v*, '*a*,'*b*,'*c*, ('*mybool*::*order*)) *model* ⇒ ('*v*, '*a*,'*b*,'*c*, '*mybool*) *model* ⇒ *bool*›
  where *leqMod* (*Da*, *Cwa*, *Fwa*, *Rwa*) (*Db*, *Cwb*, *Fwb*, *Rwb*) = ((*Da* = *Db*) ∧ *Cwa* = *Cwb* ∧ *Fwa* = *Fwb* ∧ *Rwa* ≤ *Rwb*)

lemma *leqMod-refl*: *leqMod M M* = *True*
  by (*metis* (*no-types*, *lifting*) *leqMod.simps order-refl prod-cases4*)
lemma *leqMod-antisym*: ⟦ *leqMod M1 M2* ; *leqMod M2 M1* ⟧ ⟹ *M1* = *M2*
  by (*smt order-antisym leqMod.elims*(*2*) *leqMod.simps*)
lemma *leqMod-trans*: ⟦ *leqMod M1 M2*; *leqMod M2 M3* ⟧ ⟹ *leqMod M1 M3*
  by (*smt order.trans leqMod.elims*(*2*) *leqMod.simps*)

lemma *monot-in-τ*: ‹(*DA* = *DB* ∧ *CwA* = *CwB* ∧ *FwA* = *FwB* ∧ (∀ *rsym fmlist*. *leq2* (*RwA rsym fmlist*) (*RwB rsym fmlist*)))
    ⟹ *leq2* (*value-fm-τ s* (*DA*, *CwA*, *FwA*, *RwA*) *f*) (*value-fm-τ s* (*DB*, *CwB*, *FwB*, *RwB*) *f*)›
proof(*induction f arbitrary*: *s*)
case (*Rel x1 x2*)
  then show *?case* **by** *auto*
next
  case (*Equ x1 x2*)
  then show *?case* **by** *simp*
next
  case *Fal*
  then show *?case* **by** *simp*
next
  case (*And f1 f2*)
  then show *?case* **using** *τ-and-monot* **by**(*simp add*: *less-eq-bool2-def*)
next
  case (*Neg f*)
  then show *?case* **using** *τ-not-monot* **by**(*simp add*: *less-eq-bool2-def*)
next
  case (*Forall m f*)
  then have *IH*: ∀ *s*. *value-fm-τ s* (*DA*, *CwA*, *FwA*, *RwA*) *f*
                ≤ *value-fm-τ s* (*DB*, *CwB*, *FwB*, *RwB*) *f* **by**(*simp add*: *less-eq-bool2-def*)
  fix *s*
  let *?fA* = λ *v*. *value-fm-τ* (λ *k*. *if k*=*m then v else s k*) (*DA*, *CwA*,*FwA*,*RwA*) *f*
  let *?fB* = λ *v*. *value-fm-τ* (λ *k*. *if k*=*m then v else s k*) (*DB*, *CwB*,*FwB*,*RwB*) *f*

from *IH* **have** ∀ *v*. *?fA v* ≤ *?fB v* **by** *simp* **then have** ‹*?fA* ≤ *?fB*› **by** (*simp add*: *le-funI*)
from *this τ-forall-monot Forall* **have** *τ-forall DA ?fA* ≤ *τ-forall DB ?fB* **by** *blast*

then show *leq2* (*value-fm-τ s* (*DA*, *CwA*, *FwA*, *RwA*) (*Forall m f*))
(*value-fm-τ s* (*DB*, *CwB*, *FwB*, *RwB*) (*Forall m f*)) **by**(*simp add*: *less-eq-bool2-def*)
qed

lemma *monot-in-κ*: ‹*leqMod* (*DA*, *CwA*, *FwA*, *RwA*) (*DB*, *CwB*, *FwB*, *RwB*)
⟹ *value-fm-κ s* (*DA*, *CwA*, *FwA*, *RwA*) *f* ≤ *value-fm-κ s* (*DB*, *CwB*, *FwB*, *RwB*) *f*›
proof(*induction f arbitrary*: *s*)
case (*Rel x1 x2*)
then show *?case* **by** (*simp add*: *le-funD*)
next
case (*Equ x1 x2*)
then show *?case* **by** *simp*
next
case *Fal*
then show *?case* **by** *simp*
next
case (*And f1 f2*)
then show *?case* **using** *κ-and-monot* **by** *simp*
next
case (*Neg f*)
then show *?case* **using** *κ-not-monot* **by** *simp*
next
case (*Forall m f*)
then have *IH*: ∀ *s*. *value-fm-κ s* (*DA*, *CwA*, *FwA*, *RwA*) *f*
≤ *value-fm-κ s* (*DB*, *CwB*, *FwB*, *RwB*) *f* **by** *auto*
fix *s*
let *?fA* = λ *v*. *value-fm-κ* (λ *k*. *if k=m then v else s k*) (*DA*, *CwA*,*FwA*,*RwA*) *f*
let *?fB* = λ *v*. *value-fm-κ* (λ *k*. *if k=m then v else s k*) (*DB*, *CwB*,*FwB*,*RwB*) *f*

from *Forall* **have** ‹*DA* = *DB*› **by** *auto*
from *IH* **have** ∀ *v*. *?fA v* ≤ *?fB v* **by** *simp* **then have** ‹*?fA* ≤ *?fB*› **by** (*simp add*: *le-funI*)
from *this κ-forall-monot* **have** *κ-forall DA ?fA* ≤ *κ-forall DB ?fB*
using ‹*DA* = *DB*› **by** *blast*

then show *value-fm-κ s* (*DA*, *CwA*, *FwA*, *RwA*) (*Forall m f*)
≤ *value-fm-κ s* (*DB*, *CwB*, *FwB*, *RwB*) (*Forall m f*) **by** *simp*
qed

lemma *monot-in-μ*: ‹*leqMod* (*DA*, *CwA*, *FwA*, *RwA*) (*DB*, *CwB*, *FwB*, *RwB*)
⟹ *value-fm-μ s* (*DA*, *CwA*, *FwA*, *RwA*) *f* ≤ *value-fm-μ s* (*DB*, *CwB*, *FwB*, *RwB*) *f*›
proof(*induction f arbitrary*: *s*)

case (*Rel x1 x2*)
then show *?case* **by** (*simp add*: *le-funD*)
next
 case (*Equ x1 x2*)
 then show *?case* **by** *simp*
next
case *Fal*
 then show *?case* **by** *simp*
next
 case (*And f1 f2*)
 then show *?case* **using** *μ-and-monot* **by** *simp*
next
 case (*Neg f*)
 then show *?case* **using** *μ-not-monot* **by** *simp*
next
 case (*Forall m f*)
 then have *IH*: ∀ *s*. *value-fm-μ s* (*DA*, *CwA*, *FwA*, *RwA*) *f*
   ≤ *value-fm-μ s* (*DB*, *CwB*, *FwB*, *RwB*) *f* **by** *auto*
 fix *s*
 let *?fA* = λ *v*. *value-fm-μ* (λ *k*. *if k=m then v else s k*) (*DA*, *CwA*,*FwA*,*RwA*) *f*
 let *?fB* = λ *v*. *value-fm-μ* (λ *k*. *if k=m then v else s k*) (*DB*, *CwB*,*FwB*,*RwB*) *f*

 from *Forall* **have** ‹*DA* = *DB*› **by** *auto*
 from *IH* **have** ∀ *v*. *?fA v* ≤ *?fB v* **by** *simp* **then have** ‹*?fA* ≤ *?fB*› **by** (*simp add*: *le-funI*)
 from *this μ-forall-monot* ‹*DA* = *DB*› **have** *μ-forall DA ?fA* ≤ *μ-forall DB ?fB* **by** *blast*

 then show *value-fm-μ s* (*DA*, *CwA*, *FwA*, *RwA*) (*Forall m f*)
   ≤ *value-fm-μ s* (*DB*, *CwB*, *FwB*, *RwB*) (*Forall m f*) **by** *simp*
qed

lemma *monot-in-ν*: ‹*leqMod* (*DA*, *CwA*, *FwA*, *RwA*) (*DB*, *CwB*, *FwB*, *RwB*)
 ⟹ *value-fm-ν s* (*DA*, *CwA*, *FwA*, *RwA*) *f* ≤ *value-fm-ν s* (*DB*, *CwB*, *FwB*, *RwB*) *f*›
proof(*induction f arbitrary*: *s*)
case (*Rel x1 x2*)
then show *?case* **by** (*simp add*: *le-funD*)
next
 case (*Equ x1 x2*)
 then show *?case* **by** *simp*
next
case *Fal*
 then show *?case* **by** *simp*
next
 case (*And f1 f2*)
 then show *?case* **using** *ν-and-monot* **by** *simp*
next
 case (*Neg f*)

then show *?case* using *ν-not-monot* by *simp*
next
case (*Forall m f*)
then have *IH*: ∀ *s*. *value-fm-ν s* (*DA*, *CwA*, *FwA*, *RwA*) *f*
≤ *value-fm-ν s* (*DB*, *CwB*, *FwB*, *RwB*) *f* by *auto*
fix *s*
let *?fA* = λ *v*. *value-fm-ν* (λ *k*. *if k=m then v else s k*) (*DA*, *CwA*,*FwA*,*RwA*) *f*
let *?fB* = λ *v*. *value-fm-ν* (λ *k*. *if k=m then v else s k*) (*DB*, *CwB*,*FwB*,*RwB*) *f*

from *Forall* have ‹*DA* = *DB*› by *auto*
from *IH* have ∀ *v*. *?fA v* ≤ *?fB v* by *simp* then have ‹*?fA* ≤ *?fB*› by (*simp add*: *le-funI*)
from *this ν-forall-monot*[*of ?fA ?fB*]
‹*DA* = *DB*› have *ν-forall DA ?fA* ≤ *ν-forall DB ?fB* by *simp*

then show *value-fm-ν s* (*DA*, *CwA*, *FwA*, *RwA*) (*Forall m f*)
≤ *value-fm-ν s* (*DB*, *CwB*, *FwB*, *RwB*) (*Forall m f*) by *simp*
qed

lemma *value-fm-locdet*: ‹ ⟦ ∀ *m* ∈ *freevar f*. *s1 m* = *s2 m* ⟧ ⟹
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *s1* (*D*, *Cw*, *Fw*, *Rw*) *f* =
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *s2* (*D*, *Cw*, *Fw*, *Rw*) *f*›
proof(*induction f arbitrary*: *s1 s2*)
case (*Rel symb tmlist*)
assume ∀ *m*∈*freevar* (*Rel symb tmlist*). *s1 m* = *s2 m*
let *?tmset* = *set tmlist*
have *freevar* (*Rel symb tmlist*) = ⋃ (*freevar-tm*‘ *?tmset*) by *simp*
then have ∀ *t* ∈ *?tmset*. *freevar-tm t* ⊆ *freevar* (*Rel symb tmlist*) by *auto*
then have ∀ *t* ∈ *?tmset*. *value-tm s1* (*Cw*, *Fw*) *t* = *value-tm s2* (*Cw*, *Fw*) *t*
using *value-tm-locdet* using *Rel.prems* by *blast*
then show *?case*
by (*smt map-eq-conv value-fm.simps*(*2*))
next
case (*Equ x1 x2*)
then show *?case*
by (*smt UnCI freevar.simps*(*2*) *value-fm.simps*(*3*) *value-tm-locdet*)
next
case *Fal*
then show *?case* using *value-fm.simps*(*1*) by *simp*
next
case (*And f1 f2*)
then show *?case*
by (*smt UnCI freevar.simps*(*3*) *value-fm.simps*(*4*) *value-fm-τ.simps*)
next
case (*Neg f*)
then show *?case* by *force*
next

case (Forall m f)
then have ∀ m′ ∈freevar (Forall m f). s1 m′ = s2 m′ **by** auto
then have H2: ∀ m′ ∈ freevar f. m′ ≠ m ⟶ s1 m′ = s2 m′
using freevar.simps(6) **by** simp
let ?S = (myFalse, myTrue, myNot, myAnd, myUni)

have R: ‹∀ v. value-fm ?S (λ k. if k=m then v else s1 k) (D, Cw,Fw,Rw) f
= value-fm ?S (λ k. if k=m then v else s2 k) (D, Cw,Fw,Rw) f›
proof
fix v **let** ?s1 = (λ k. if k=m then v else s1 k) **let** ?s2 = (λ k. if k=m then v else s2 k)
have (∀ m∈freevar f. ?s1 m = ?s2 m) **using** H2 **by** simp
from this **have** value-fm ?S ?s1 (D, Cw, Fw, Rw) f = value-fm ?S ?s2 (D, Cw, Fw, Rw) f **by**(simp add: Forall.IH)
then show ‹value-fm ?S (λ k. if k=m then v else s1 k) (D, Cw,Fw,Rw) f = value-fm ?S (λ k. if k=m then v else s2 k) (D, Cw,Fw,Rw) f› **by** auto
qed

from R **show** value-fm ?S s1 (D, Cw, Fw, Rw) (Forall m f) =
value-fm ?S s2 (D, Cw, Fw, Rw) (Forall m f) **by** simp
qed

lemma value-fm-locdetS: ‹ ⟦ ∀ m ∈ freevar f. s1 m = s2 m ⟧ ⟹
value-fm S s1 (D, Cw, Fw, Rw) f = value-fm S s2 (D, Cw, Fw, Rw) f›
using value-fm-locdet **by** (smt prod-cases5)

fun nthelement-in-set ::‹nat set ⇒ nat ⇒ nat› **where**
nthelement-in-set S 0 = Min S |
nthelement-in-set S (Suc n) = Min { s ∈ S . (nthelement-in-set S n) < s }

lemma value-fm-locdetS-cont: ‹ ⟦ ∀ m ∈ contvar f. s1 m = s2 m ⟧ ⟹
value-fm S s1 (D, Cw, Fw, Rw) f = value-fm S s2 (D, Cw, Fw, Rw) f›
using value-fm-locdetS freevar-contvar **by** blast

fun pos-in-list::‹′a list ⇒ ′a ⇒ (nat × bool)› **where**
‹pos-in-list L a = (fold (λ a′ (c, b). if(¬ b) then (if(a′=a) then (c, True) else (c+1, False))
else (c,b)) L (0, False))›

fun manufactured-assignment :: ‹nat list ⇒ ′v list ⇒ ′v assignment› **where**
‹manufactured-assignment varList valList =
(λ M. if (M ∈ set varList)
then valList ! fst (pos-in-list (sort varList) M) else undefined) ›

fun value-fm′ :: ‹(′v, ′mybool) scheme ⇒ ′v list ⇒ (′v, ′a, ′b, ′c, ′mybool) model

$\Rightarrow$ ('a, 'b, 'c) *fm* $\Rightarrow$ '*mybool*› **where**
value-fm' S val-list (*D*, *Cw*, *Fw*, *Rw*) *f*
= (*if* (*length val-list* = *length* (*freevarL f*))
then value-fm S (*manufactured-assignment* (*freevarL f*) *val-list*)
(*D*, *Cw*, *Fw*, *Rw*) *f else undefined*)

lemma *value-fm'-test1*:
value-fm' τ [*2*, *2*, *1*::*nat*] (*D*, *Cw*, *Fw*, *Rw*) (*And* (*Equ* (*Var 10*) (*Var 2*))
(*Equ* (*Var 10*) (*Var 3*))) = *f2* **by** *simp*

lemma *value-fm'-test2*:
value-fm' τ [*2*, *2*, *2*::*nat*] (*D*, *Cw*, *Fw*, *Rw*) (*And* (*Equ* (*Var 10*) (*Var 2*))
(*Equ* (*Var 10*) (*Var 3*))) = *t2* **by** *simp*

fun *signification* :: ‹('v, '*mybool*) *scheme*
$\Rightarrow$ ('v, 'a, 'b, 'c, '*mybool*) *model* $\Rightarrow$ ('a, 'b, 'c) *fm*
$\Rightarrow$ ('*v list* $\Rightarrow$ '*mybool*)› **where**
signification S Mod fm = (λ *value-list*. *value-fm' S value-list Mod fm*)

fun *extension* :: ‹ ('v, ('*mybool* :: *order*)) *scheme*
$\Rightarrow$ ('v, 'a, 'b, 'c, '*mybool*) *model* $\Rightarrow$ ('a, 'b, 'c) *fm*
$\Rightarrow$ ('*v list set*)› **where**
extension (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *Mod fm*
= {*vl* :: '*v list*. *myTrue* $\leq$ (*signification* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *Mod fm vl*)}

fun *antiextension* :: ‹ ('v, ('*mybool* :: *order*)) *scheme*
$\Rightarrow$ ('v, 'a, 'b, 'c, '*mybool*) *model* $\Rightarrow$ ('a, 'b, 'c) *fm*
$\Rightarrow$ ('*v list set*)› **where**
antiextension (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *Mod fm*
= {*vl* :: '*v list*. *myFalse* $\leq$ (*signification* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *Mod fm vl*)}

3.1 Substitutions

subst_term tm n t1 $\equiv$ in the term tm replace all occurences of variable n with the term t1

fun *subst-term* :: ('a, 'b) *tm* $\Rightarrow$ *nat* $\Rightarrow$ ('a, 'b) *tm* $\Rightarrow$ ('a, 'b) *tm* **where**
(*subst-term* (*Var n*) (*m* ::*nat*) *t1*) = (*if n* = *m then t1 else* (*Var n*)) |
(*subst-term* (*Const c*) *m t1*) = (*Const c*) |
(*subst-term* (*Fun f-symb term-list*) *m t1*) =
(*Fun f-symb* ((*map* (λ *x*. *subst-term x m t1*) *term-list*)))

fun *subst-termS* :: ('a, 'b) *tm* $\Rightarrow$ (*nat* $\Rightarrow$ ('a, 'b) *tm*) $\Rightarrow$ ('a, 'b) *tm* **where**
subst-termS (*Var n*) ϑ = ϑ *n* |
subst-termS (*Const c*) ϑ = (*Const c*) |
subst-termS (*Fun f-symb term-list*) ϑ =
(*Fun f-symb* ((*map* (λ *x*. *subst-termS x* ϑ) *term-list*)))

abbreviation ι:: *nat* ⇒ ('*a*, '*b*) *tm* **where**
‹$\iota \equiv (\lambda\ n.\ Var\ n)$›

lemma *subst-termS-test*: $subst\text{-}termS\ A\ (\iota\ (n := t)) = subst\text{-}term\ A\ n\ t$
  by(*induction A*; *simp*)

lemma *subst-term-test*:
‹*subst-term* (*Fun f* [*Const c1*, *Var 1*, *Var 3*]) *3* (*Const c2*)
  = (*Fun f* [*Const c1*, *Var 1*, *Const c2*])› **by** *simp*

lemma *subst-term-on-closedt-is-id*:
$freevar\text{-}tm\ tm = \{\} \Longrightarrow (subst\text{-}term\ tm\ x\ t) = tm$
  by(*induction tm*; *simp add*:*map-idI*)

lemma *subst-term-is-id2*:
$x \notin freevar\text{-}tm\ tm \Longrightarrow (subst\text{-}term\ tm\ x\ t) = tm$
  by(*induction tm*; *simp add*:*map-idI*)

fun *updt-w-subst* :: ‹(('*v*, '*a*,'*b*,'*c*) *const-mod* × ('*v*, '*a*,'*b*,'*c*) *func-mod*)
  ⇒ '*v assignment* ⇒ (*nat* ⇒ ('*a*, '*b*) *tm*) ⇒ '*v assignment*› **where**
$updt\text{-}w\text{-}subst\ (Cw,\ Fw)\ w\ \vartheta\ = (\lambda\ m.\ value\text{-}tm\ w\ (Cw,\ Fw)\ (\vartheta\ m))$

lemma *updt-w-subst-is-id*:
‹ $\vartheta\ m = Var\ m \Longrightarrow (updt\text{-}w\text{-}subst\ (Cw,\ Fw)\ w\ \vartheta)\ m = w\ m$› **by** *simp*

lemma *substitution-theoremT*:
‹$value\text{-}tm\ w\ (Cw,\ Fw)\ (subst\text{-}termS\ t\ \vartheta)$
  $= value\text{-}tm\ (updt\text{-}w\text{-}subst\ (Cw,\ Fw)\ w\ \vartheta)\ (Cw,\ Fw)\ t$›
  apply(*induction t*; *simp*) **by** (*smt comp-apply map-eq-conv*)

fun *substu* **where**
substu x t f =
  ($if\ (x \notin freevar\text{-}tm\ t)\ then\ x$
    $else\ Min\ \{v.\ (v \notin freevar\text{-}tm\ t) \wedge (v \notin contvar\ f)\}$)

fun *qtrg* :: ‹('*a*, '*b*, '*c*) *fm* ⇒ *nat*› **where**
qtrg (*Rel Rs tml*) = *0* |
qtrg (*Equ tm1 tm2*) = *0* |
qtrg (*And fm1 fm2*) = *Max* {*qtrg fm1*, *qtrg fm2*} +*1* |
qtrg (*Neg fm1*) = *qtrg fm1* +*1* |
qtrg Fal = *0* |
qtrg (*Forall var A*) = *qtrg A* +*10*

abbreviation *do-sub* **where**
$do\text{-}sub\ N\ x\ phi\ \vartheta \equiv (N \in freevar\ (Forall\ x\ phi) \wedge Var\ N \neq \vartheta\ N)$

fun *substuS* **where**
$substuS\ x\ phi\ \vartheta =$
  ($if\ (\forall\ m.\ (do\text{-}sub\ m\ x\ phi\ \vartheta) \longrightarrow x \notin freevar\text{-}tm\ (\vartheta\ m))\ then\ x$

else Inf $\{v.\ (\forall\ m.\ (\textit{do-sub}\ m\ x\ \textit{phi}\ \vartheta) \longrightarrow v \notin \textit{freevar-tm}\ (\vartheta\ m))$
$\wedge\ (v \notin \textit{contvar phi})\})$

abbreviation *bounded* **where**
bounded $\vartheta \equiv (\exists\ n.\ \forall\ N > n.\ \vartheta\ N = \textit{Var}\ N)$

lemma *finite-freevar-tm*:‹*finite* (*freevar-tm t*)›
by(*induction t*; *simp*)

lemma *finite-contvar*:‹*finite* (*contvar phi*)›
by(*induction phi*; *simp*; *metis List.finite-set freevar-tm-id*)

lemma *finite-bounded*: ‹$[\![$ *finite* (*A* :: *nat set*) $]\!]$
$\Longrightarrow \exists\ n.\ \forall\ N > n.\ N \notin A$›
proof(*cases* $A = \{\}$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
assume ‹*finite A*›
from *this False* **have** $\forall\ N > \textit{Max}\ A.\ N \notin A$ **by** *auto*
then show *?thesis* **by**(*rule exI*)
qed

lemma *bounded-contvar*:‹$\exists\ n.\ \forall\ N > n.\ N \notin$ (*contvar phi*)›
using *finite-contvar finite-bounded* **by** *auto*

lemma *bounded-finite*: ‹$[\![\ \exists\ n.\ \forall\ N > n.\ N \notin (A :: \textit{nat set})\ ]\!]$
$\Longrightarrow$ *finite A*›
proof−
fix *A* :: ‹*nat set*›
assume *H*: $\exists\ n.\ \forall\ N > n.\ N \notin A$
then obtain *n* **where** *Hn*: ‹$\forall\ N > n.\ N \notin A$› **by** *auto*

have *A*: *finite* $\{n'.\ n' \le n\}$ **using** *finite-Collect-le-nat* **by** *simp*
have *B*: $A \subseteq \{n'.\ n' \le n\}$ **using** *Hn*
by (*metis mem-Collect-eq not-le-imp-less subsetI*)

from *A B* **show** *finite A* **using** *finite-subset* **by** *auto*
qed

lemma *substuS-notin-freevar-tm*:
‹$[\![$ *bounded* ϑ; *do-sub m x phi* ϑ $]\!]$
$\Longrightarrow$ (*substuS x phi* ϑ) $\notin$ *freevar-tm* (ϑ *m*)›
proof(*cases* $(\forall\ m.\ (\textit{do-sub}\ m\ x\ \textit{phi}\ \vartheta) \longrightarrow x \notin \textit{freevar-tm}\ (\vartheta\ m)))$
case *True*
then have *I*: *substuS x phi* $\vartheta = x$ **by** *simp*
assume *do-sub m x phi* ϑ
then have $x \notin$ *freevar-tm* (ϑ *m*) **using** *True* **by** *blast*

then show (*substuS x phi* ϑ) ∉ *freevar-tm* (ϑ *m*) **using** *I* **by** *simp*
next
case *False*
let *?newu* = *Inf* {*v*. (∀ *m*. (*do-sub m x phi* ϑ) ⟶ *v* ∉ *freevar-tm* (ϑ *m*))
∧ (*v* ∉ *contvar phi*)}
assume ‹*bounded* ϑ›
then have *A*: *finite* {*m*. ϑ *m* ≠ ι *m*} **using** *bounded-finite* **by** *simp*
have ‹{*m*. *do-sub m x phi* ϑ} ⊆ {*m*. ϑ *m* ≠ ι *m*}› **by** *auto*
then have *A′*: ‹*finite* {*m*. *do-sub m x phi* ϑ}›
using *A finite-subset* **by** *auto*

have *B*: ∀ *m* . *finite* (*freevar-tm* (ϑ *m*)) **using** *finite-freevar-tm* **by** *auto*

let *?S* = ‹{*m′*. ∃ *m*. (*do-sub m x phi* ϑ) ∧ *m′* ∈ *freevar-tm* (ϑ *m*)}›
have ‹*?S* = ⋃ { *freevar-tm* (ϑ *m*) | *m*. *do-sub m x phi* ϑ}› **by** *auto*
from *this A′ B* **have** ‹*finite ?S*›
using *finite-Union* **by** *auto*
hence ‹*finite* {*m′*. ¬ (∀ *m*. (*do-sub m x phi* ϑ) ⟶ *m′* ∉ *freevar-tm* (ϑ *m*))}› **by** *simp*
hence
∃ *n*. ∀ *N* > *n*. *N* ∉ {*m′*. ¬ (∀ *m*. (*do-sub m x phi* ϑ) ⟶ *m′* ∉ *freevar-tm* (ϑ *m*))}
by(*rule finite-bounded*)
then obtain *n1* **where** *Hn1*:
∀ *N* > *n1*. (∀ *m*. (*do-sub m x phi* ϑ) ⟶ *N* ∉ *freevar-tm* (ϑ *m*))
by *auto*

from *bounded-contvar* **have** ∃ *n2*. ∀ *N* > *n2*. *N* ∉ *contvar phi* **by** *auto*
then obtain *n2* **where** *Hn2*: ∀ *N* > *n2*. *N* ∉ *contvar phi* **by** *auto*

let *?N* = *Max*{*n1*, *n2*} +*1*
from *Hn1* **have** *A*: ‹(∀ *m*. (*do-sub m x phi* ϑ) ⟶ *?N* ∉ *freevar-tm* (ϑ *m*))› **by** *auto*
from *Hn2* **have** *B*: ‹*?N* ∉ *contvar phi*› **by** *auto*

from *Hn1 Hn2* **have** *?N* ∈ {*v*. (∀ *m*. (*do-sub m x phi* ϑ) ⟶ *v* ∉ *freevar-tm* (ϑ *m*))
∧ (*v* ∉ *contvar phi*)} **by** *simp*
then have {*v*. (∀ *m*. (*do-sub m x phi* ϑ) ⟶ *v* ∉ *freevar-tm* (ϑ *m*))
∧ (*v* ∉ *contvar phi*)} ≠ {} **by** *auto*
then have *P*: (∀ *m*. (*do-sub m x phi* ϑ) ⟶ *?newu* ∉ *freevar-tm* (ϑ *m*))
∧ (*?newu* ∉ *contvar phi*) **using** *Inf-nat-def1*
by (*smt mem-Collect-eq*)
assume *do-sub m x phi* ϑ
then have *?newu* ∉ *freevar-tm* (ϑ *m*) **using** *P* **by** *simp*
then show *?thesis* **using** *False* **by** *auto*
qed

lemma *substuSι-is-id*: *substuS x phi* ι = *x* **by** *simp*

```
lemma substuSxx-is-id: substuS x phi (ι (x := tm)) = x by simp

fun substϑS where
substϑS x phi ϑ = ( λ N. if(do-sub N x phi ϑ) then ϑ N
   else if N = x then Var (substuS x phi ϑ) else Var N)

lemma substϑSxx-is-id: substϑS x phi (ι (x := tm)) = ι
  by( simp add:substuSxx-is-id; auto)

lemma substϑSι-is-id: substϑS x phi ι = ι
  by(simp add: substuSι-is-id; auto)

fun subst-formS :: ('a, 'b, 'c) fm ⇒ (nat ⇒ ('a, 'b) tm) ⇒ ('a, 'b, 'c) fm where
subst-formS (Rel Rsymb tml) ϑ = Rel Rsymb (map (λ x. subst-termS x ϑ) tml ) |
subst-formS (Equ tm1 tm2) ϑ = Equ (subst-termS tm1 ϑ) (subst-termS tm2 ϑ) |
subst-formS Fal ϑ = Fal |
subst-formS (And fm1 fm2) ϑ = And (subst-formS fm1 ϑ) (subst-formS fm2 ϑ) |
subst-formS (Neg f) ϑ = Neg (subst-formS f ϑ) |
subst-formS (Forall x phi) ϑ = Forall (substuS x phi ϑ)
       (subst-formS phi (substϑS x phi ϑ) )

lemma subst-termS-ι-is-id: (subst-termS tm ι) = tm
  by(induction tm; simp add: map-idI)

lemma subst-formS-ι-is-id:(subst-formS phi ι) = phi
proof(induction phi)
case (Rel x1 x2)
  then show ?case by(simp add: subst-termS-ι-is-id)
next
case (Equ x1 x2)
  then show ?case by(simp add: subst-termS-ι-is-id)
next
  case Fal
  then show ?case by auto
next
  case (And phi1 phi2)
  then show ?case by auto
next
  case (Neg phi)
  then show ?case by auto
next
  case (Forall x1 phi)
  have subst-formS (Forall x1 phi) ι = Forall x1
       (subst-formS phi (substϑS x1 phi ι) ) by simp
  then have subst-formS (Forall x1 phi) ι = Forall x1
       (subst-formS phi ι ) using substϑSι-is-id by (metis)
  then show ?case using Forall by simp
qed
```

lemma *substϑS-bounded*:
bounded $\vartheta \Longrightarrow$ *bounded* (*substϑS var fm* ϑ)
proof−
  assume ‹*bounded* ϑ›
  then obtain n **where** $\forall N>n.\ \vartheta\ N = \iota\ N$ **by** *auto*
  then have *Hn*: $\forall\ N >\ n.$ *do-sub N var fm* ϑ = *False* **by** *auto*
  let $?n' = Max\ \{var + 1,\ n\}$
  from *Hn* **have** *Hn'*: ‹$\forall\ N >\ ?n'.$ *do-sub N var fm* ϑ = *False* $\wedge$ $N \neq var$› **by** *auto*
  hence $\forall\ N >\ ?n'.$ *substϑS var fm* $\vartheta\ N = \iota\ N$ **by** *simp*
  thus *bounded* (*substϑS var fm* ϑ) **by**(*rule exI*)
qed

lemma *on-var-dont-sub*:
  do-sub m var fm $\vartheta \Longrightarrow m \neq var$ **by** *simp*

lemma *substitution-theoremF*:
‹ *bounded* $\vartheta \Longrightarrow$
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *w* (*D*, *Cw*, *Fw*, *Rw*)
  (*subst-formS fm* ϑ) = *value-fm* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*)
  (*updt-w-subst* (*Cw*, *Fw*) *w* ϑ) (*D*, *Cw*, *Fw*, *Rw*) *fm*›
proof(*induction fm arbitrary*: *w* ϑ)
let *?S* = (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*)
  case (*Rel Rsymb tmlist*)
  have ‹$\forall$ *tm* $\in$ *set tmlist*.
  (*value-tm* ($\lambda m.$ *value-tm w* (*Cw*, *Fw*) (ϑ *m*)) (*Cw*, *Fw*) *tm*
    = (*value-tm w* (*Cw*, *Fw*)) (*subst-termS tm* ϑ))› **by** (*simp add*: *substitution-theoremT*)
  then have *Hi*: (*map* (*value-tm w* (*Cw*, *Fw*)) (*map* ($\lambda x.$ *subst-termS x* ϑ) *tmlist*)) =
(*map* (*value-tm* ($\lambda m.$ *value-tm w* (*Cw*, *Fw*) (ϑ *m*)) (*Cw*, *Fw*)) *tmlist*) **by** *simp*

  have ‹*value-fm ?S w* (*D*, *Cw*, *Fw*, *Rw*) (*subst-formS* (*Rel Rsymb tmlist*) ϑ)
    = *value-fm ?S w* (*D*, *Cw*, *Fw*, *Rw*) (*Rel Rsymb* (*map* ($\lambda x.$ *subst-termS x* ϑ) *tmlist*))› **by** *simp*
  also have ... = *Rw Rsymb* (*map* (*value-tm w* (*Cw*, *Fw*))
    (*map* ($\lambda x.$ *subst-termS x* ϑ) *tmlist*)) **by** *simp*
  also have ... = *Rw Rsymb* (*map* (*value-tm* ($\lambda m.$ *value-tm w* (*Cw*, *Fw*) (ϑ *m*)) (*Cw*, *Fw*)) *tmlist*)
    using *Hi* **by** *metis*
  also have ... = *value-fm ?S* (*updt-w-subst* (*Cw*, *Fw*) *w* ϑ) (*D*, *Cw*, *Fw*, *Rw*) (*Rel Rsymb tmlist*) **by** *simp*
  finally show *?case* **by** *simp*
next
case (*Equ x1 x2*)
  then show *?case* **using** *substitution-theoremT*
    subst-formS.simps(*2*) *value-fm.simps*(*3*) **by** *smt*
next
  case *Fal*
  then show *?case* **by** *simp*

next
  case (*And fm1 fm2*)
  then show *?case* **by** *simp*
next
  case (*Neg fm*)
  then show *?case* **by** *simp*
next
  case (*Forall var fm*)
  let *?S* = (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*)
  from *Forall.IH* **have** *IH*: $\bigwedge$ *w* ϑ. (*bounded* ϑ) $\Longrightarrow$
  value-fm ?S w (*D*, *Cw*, *Fw*, *Rw*) (*subst-formS fm* ϑ) =
  value-fm ?S (*updt-w-subst* (*Cw*, *Fw*) *w* ϑ) (*D*, *Cw*, *Fw*, *Rw*) *fm* **by** *simp*

  fix *w*
  fix ϑ :: ‹*nat* ⇒ ('*a*,'*b*) *tm*›
  assume ‹*bounded* ϑ›
  hence *Hbsubst*ϑ: *bounded* (*subst*ϑ*S var fm* ϑ) **by**(*rule subst*ϑ*S-bounded*)
  have *value-fm ?S w* (*D*, *Cw*, *Fw*, *Rw*) (*subst-formS* (*Forall var fm*) ϑ)
      = *value-fm ?S w* (*D*, *Cw*, *Fw*, *Rw*) (*Forall* (*substuS var fm* ϑ)
   (*subst-formS fm* (*subst*ϑ*S var fm* ϑ))) **by** *simp*
  moreover have ... = *myUni D* (λ *v*. *value-fm ?S* (λ*k*. *if k* = (*substuS var fm*
ϑ)
  then v else w k) (*D*, *Cw*, *Fw*, *Rw*) (*subst-formS fm* (*subst*ϑ*S var fm* ϑ))) **by** *simp*
  moreover have ... = *myUni D* (λ *v*. *value-fm ?S* (*updt-w-subst* (*Cw*, *Fw*)
   (λ*k*. *if k* = (*substuS var fm* ϑ) *then v else w k*)
    (*subst*ϑ*S var fm* ϑ)) (*D*, *Cw*, *Fw*, *Rw*) *fm*) **using** *IH Hbsubst*ϑ **by** *simp*
  moreover have ... = *myUni D* (λ *v*. *value-fm ?S* (λ *m*. *value-tm*
   (λ*k*. *if k* = (*substuS var fm* ϑ) *then v else w k*) (*Cw*, *Fw*)
    (*subst*ϑ*S var fm* ϑ *m*)) (*D*, *Cw*, *Fw*, *Rw*) *fm*) **by** *simp*
  ultimately have *IdL*: ‹*value-fm ?S w* (*D*, *Cw*, *Fw*, *Rw*) (*subst-formS* (*Forall*
var fm) ϑ)
  = *myUni D* (λ *v*. *value-fm ?S* (λ *m*. *value-tm*
   (λ*k*. *if k* = (*substuS var fm* ϑ) *then v else w k*) (*Cw*, *Fw*)
    (*subst*ϑ*S var fm* ϑ *m*)) (*D*, *Cw*, *Fw*, *Rw*) *fm*)› **by** *simp*

  have *IdM*: $\bigwedge$ *v*. *value-fm ?S* (λ *m*. *value-tm*
   (λ*k*. *if k* = (*substuS var fm* ϑ) *then v else w k*) (*Cw*, *Fw*)
    (*subst*ϑ*S var fm* ϑ *m*)) (*D*, *Cw*, *Fw*, *Rw*) *fm*
 = *value-fm ?S* (λ *m*. (*if m* $\neq$ *var then value-tm*
     w (*Cw*, *Fw*) (*subst*ϑ*S var fm* ϑ *m*) *else v*))
    (*D*, *Cw*, *Fw*, *Rw*) *fm* **proof**−
  fix *a*
  let *?ws* = (λ*k*. *if k* = (*substuS var fm* ϑ) *then a else w k*)

  have ∀ *m*. *value-tm ?ws* (*Cw*, *Fw*) (*subst*ϑ*S var fm* ϑ *m*)
   = (*if*(*m* $\neq$ *var*) *then value-tm ?ws* (*Cw*, *Fw*) (*subst*ϑ*S var fm* ϑ *m*)
    else a) **by** *auto*
  hence *Calc1*: *value-fm ?S* (λ *m*. *value-tm ?ws* (*Cw*, *Fw*)
   (*subst*ϑ*S var fm* ϑ *m*)) (*D*, *Cw*, *Fw*, *Rw*) *fm*

$= value\text{-}fm\ ?S\ (\lambda\ m.\ (if\ m \neq var\ then\ value\text{-}tm$
$?ws\ (Cw,\ Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m)\ else\ a))$
$(D,\ Cw,\ Fw,\ Rw)\ fm$ **by** *simp*

let $?w1 = (\lambda\ m.\ (if\ m \neq var\ then\ value\text{-}tm$
$?ws\ (Cw,\ Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m)\ else\ a))$
let $?w2 = (\lambda\ m.\ (if\ m \neq var\ then\ value\text{-}tm$
$w\ (Cw,\ Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m)\ else\ a))$

have $\forall\, m.\ do\text{-}sub\ m\ var\ fm\ \vartheta \longrightarrow$
$value\text{-}tm\ ?ws\ (Cw,\ Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m) =$
$value\text{-}tm\ w\ (Cw,\ Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m)$
proof
fix m
show $do\text{-}sub\ m\ var\ fm\ \vartheta \longrightarrow value\text{-}tm\ ?ws\ (Cw,\ Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m) = value\text{-}tm\ w\ (Cw,\ Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m)$ **proof**
assume $do\text{-}sub\ m\ var\ fm\ \vartheta$
hence *Hu*: $substuS\ var\ fm\ \vartheta \notin freevar\text{-}tm\ (\vartheta\ m)$ **using** $\langle bounded\ \vartheta\rangle$ **by**(*metis substuS-notin-freevar-tm*)
have $\bigwedge\ m'.\ ?ws\ m' \neq w\ m' \Longrightarrow m' = (substuS\ var\ fm\ \vartheta)$ **by** *metis*
hence *Coinc-loc*:
$\bigwedge\ m'.\ m' \neq (substuS\ var\ fm\ \vartheta) \Longrightarrow ?ws\ m' = w\ m'$ **by** *metis*
have $\bigwedge\ m'.\ m' \in freevar\text{-}tm\ (\vartheta\ m) \Longrightarrow m' \neq (substuS\ var\ fm\ \vartheta)$ **using** *Hu* **by** *blast*
hence $\bigwedge\ m'.\ m' \in freevar\text{-}tm\ (\vartheta\ m) \Longrightarrow ?ws\ m' = w\ m'$ **using** *Coinc-loc* **by** *auto*
then have $\forall\, m'{\in}freevar\text{-}tm\ (\vartheta\ m).\ ?ws\ m' = w\ m'$ **by** *metis*
thus $value\text{-}tm\ ?ws\ (Cw,\ Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m) =$
$value\text{-}tm\ w\ (Cw,\ Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m)$
using $value\text{-}tm\text{-}locdet[of\ \vartheta\ m\ ?ws\ w\ Cw\ Fw]$
using $\langle do\text{-}sub\ m\ var\ fm\ \vartheta\rangle$ **by** *auto*
qed
qed
from *this* **have** *Coinc1*:
$\forall\, m.\ do\text{-}sub\ m\ var\ fm\ \vartheta \longrightarrow ?w1\ m = ?w2\ m$ **by** *simp*
have *Coinc2*: $?w1\ var = ?w2\ var$ **by** *simp*

have *A*: $\forall\, m.\ \neg\ do\text{-}sub\ m\ var\ fm\ \vartheta \longrightarrow m \neq var$
$\longrightarrow ?w1\ m = ?ws\ m$ **by** *simp*
have *B*: $\forall\, m.\ \neg\ do\text{-}sub\ m\ var\ fm\ \vartheta \longrightarrow m \neq var$
$\longrightarrow ?w2\ m = w\ m$ **by** *simp*
have *C*: $\forall\, m.\ w\ m \neq ?ws\ m \longrightarrow m = substuS\ var\ fm\ \vartheta$ **by** *simp*

have $\forall\ m.\ \neg\ do\text{-}sub\ m\ var\ fm\ \vartheta \longrightarrow m \neq var$
$\longrightarrow ?w1\ m \neq ?w2\ m \longrightarrow m{=}substuS\ var\ fm\ \vartheta$
proof
fix m **show** $\neg\ do\text{-}sub\ m\ var\ fm\ \vartheta \longrightarrow m \neq var$
$\longrightarrow ?w1\ m \neq ?w2\ m \longrightarrow m{=}substuS\ var\ fm\ \vartheta$
proof assume *H1*: $\neg\ do\text{-}sub\ m\ var\ fm\ \vartheta$ **show**

$m \neq var \longrightarrow ?w1\ m \neq ?w2\ m \longrightarrow m=substuS\ var\ fm\ \vartheta$ **proof**
  assume $H2$: $m \neq var$ **show** $?w1\ m \neq ?w2\ m \longrightarrow m=substuS\ var\ fm\ \vartheta$ **proof**
    assume $H3$: $?w1\ m \neq ?w2\ m$
    from $H1\ H2\ A$ **have** D: $?w1\ m = ?ws\ m$ **by** *simp*
    from $H1\ H2\ B$ **have** E: $?w2\ m = w\ m$ **by** *simp*
    from $D\ E\ H3$ **have** $\langle ?ws\ m \neq w\ m\rangle$ **by** *simp*
    then have F: $w\ m \neq ?ws\ m$ **by**(*rule not-sym*)
    from C **have** $w\ m \neq (\textit{if}\ m = substuS\ var\ fm\ \vartheta$
        $\textit{then}\ a\ \textit{else}\ w\ m) \longrightarrow m = substuS\ var\ fm\ \vartheta$ **by**(*rule allE*)
    from F *this* **show** $m=substuS\ var\ fm\ \vartheta$ **by** *auto*
  qed
 qed
qed
qed
  then have $Coinc3$: $\forall\ m.\ \neg\ do\text{-}sub\ m\ var\ fm\ \vartheta \longrightarrow m \neq var$
  $\longrightarrow m \neq substuS\ var\ fm\ \vartheta \longrightarrow ?w1\ m = ?w2\ m$ **by** *simp*

  from $Coinc1\ Coinc2\ Coinc3$ **have** $Coinc$: $\forall\ m.$
    $m \neq substuS\ var\ fm\ \vartheta \longrightarrow ?w1\ m = ?w2\ m$ **by** *blast*

  have $Fact$: $substuS\ var\ fm\ \vartheta \neq var \Longrightarrow$
      $substuS\ var\ fm\ \vartheta \notin contvar\ fm$
  proof−
    assume H: $substuS\ var\ fm\ \vartheta \neq var$
    let $?newu = Inf\ \{v.\ (\forall\ m.\ (do\text{-}sub\ m\ var\ fm\ \vartheta) \longrightarrow v \notin freevar\text{-}tm\ (\vartheta\ m))$
        $\wedge\ (v \notin contvar\ fm)\}$
    from H **have** $Id1$: $substuS\ var\ fm\ \vartheta = ?newu$ **by** *auto*

    from $\langle bounded\ \vartheta\rangle$ **have** A: $finite\ \{m.\ \vartheta\ m \neq \iota\ m\}$ **using** *bounded-finite* **by** *simp*
    have $\langle\{m.\ do\text{-}sub\ m\ var\ fm\ \vartheta\} \subseteq \{m.\ \vartheta\ m \neq \iota\ m\}\rangle$ **by** *auto*
    then have A': $\langle finite\ \{m.\ do\text{-}sub\ m\ var\ fm\ \vartheta\}\rangle$
    using A *finite-subset* **by** *auto*

    have B: $\forall\ m\ .\ finite\ (freevar\text{-}tm\ (\vartheta\ m))$ **using** *finite-freevar-tm* **by** *auto*

    let $?S = \langle\{m'.\ \exists\ m.\ (do\text{-}sub\ m\ var\ fm\ \vartheta) \wedge m' \in freevar\text{-}tm\ (\vartheta\ m)\}\rangle$
    have $\langle ?S = \bigcup\ \{\ freevar\text{-}tm\ (\vartheta\ m)\ |\ m.\ do\text{-}sub\ m\ var\ fm\ \vartheta\}\rangle$ **by** *auto*
    from *this* $A'\ B$ **have** $\langle finite\ ?S\rangle$
    using *finite-Union* **by** *auto*
    hence $\langle finite\ \{m'.\ \neg\ (\forall\ m.\ (do\text{-}sub\ m\ var\ fm\ \vartheta) \longrightarrow m' \notin freevar\text{-}tm\ (\vartheta\ m))\}\rangle$
by *simp*
    hence
      $\exists\ n.\ \forall\ N > n.\ N \notin \{m'.\ \neg\ (\forall\ m.\ (do\text{-}sub\ m\ var\ fm\ \vartheta) \longrightarrow m' \notin freevar\text{-}tm$
$(\vartheta\ m))\}$
      by(*rule finite-bounded*)
    then obtain $n1$ **where** $Hn1$:
      $\forall\ N > n1.\ (\forall\ m.\ (do\text{-}sub\ m\ var\ fm\ \vartheta) \longrightarrow N \notin freevar\text{-}tm\ (\vartheta\ m))$
      by *auto*

from *bounded-contvar* **have** $\exists\ n2.\ \forall\ N > n2.\ N \notin contvar\ fm$ **by** *auto*
then obtain *n2* **where** *Hn2*: $\forall\ N > n2.\ N \notin contvar\ fm$ **by** *auto*

let $?N = Max\{n1, n2\} + 1$
from *Hn1* **have** *A*: $\langle(\forall\ m.\ (do\text{-}sub\ m\ var\ fm\ \vartheta) \longrightarrow ?N \notin freevar\text{-}tm\ (\vartheta\ m))\rangle$ **by** *auto*
from *Hn2* **have** *B*: $\langle ?N \notin contvar\ fm\rangle$ **by** *auto*

from *Hn1 Hn2* **have** $?N \in \{v.\ (\forall\ m.\ (do\text{-}sub\ m\ var\ fm\ \vartheta) \longrightarrow v \notin freevar\text{-}tm\ (\vartheta\ m))$
$\wedge\ (v \notin contvar\ fm)\}$ **by** *simp*
then have $\{v.\ (\forall\ m.\ (do\text{-}sub\ m\ var\ fm\ \vartheta) \longrightarrow v \notin freevar\text{-}tm\ (\vartheta\ m))$
$\wedge\ (v \notin contvar\ fm)\} \neq \{\}$ **by** *auto*
then have *P*: $(\forall\ m.\ (do\text{-}sub\ m\ var\ fm\ \vartheta) \longrightarrow ?newu \notin freevar\text{-}tm\ (\vartheta\ m))$
$\wedge\ (?newu \notin contvar\ fm)$ **using** *Inf-nat-def1*
by (*smt mem-Collect-eq*)
then have $?newu \notin contvar\ fm$ **using** *P* **by** *simp*
thus $substuS\ var\ fm\ \vartheta \notin contvar\ fm$ **using** *Id1* **by** *auto*
qed

have *P1*: $substuS\ var\ fm\ \vartheta \neq var \Longrightarrow$
$value\text{-}fm\ ?S\ ?w1\ (D, Cw, Fw, Rw)\ fm = value\text{-}fm\ ?S\ ?w2\ (D, Cw, Fw, Rw)\ fm$
proof−
assume *H*: $substuS\ var\ fm\ \vartheta \neq var$
have $\forall m{\in}contvar\ fm.\ ?w1\ m = ?w2\ m$ **proof**
fix *m*
assume $m \in contvar\ fm$
from *this Fact H* **have** $m \neq substuS\ var\ fm\ \vartheta$ **by** *blast*
then show $?w1\ m = ?w2\ m$ **using** *Coinc* **by** *simp*
qed
then show $value\text{-}fm\ ?S\ ?w1\ (D, Cw, Fw, Rw)\ fm = value\text{-}fm\ ?S\ ?w2\ (D, Cw, Fw, Rw)\ fm$
using *value-fm-locdetS-cont*[*of fm ?w1 ?w2*] **by** *simp*
qed

have *P2*: $substuS\ var\ fm\ \vartheta = var \Longrightarrow$
$value\text{-}fm\ ?S\ ?w1\ (D, Cw, Fw, Rw)\ fm = value\text{-}fm\ ?S\ ?w2\ (D, Cw, Fw, Rw)\ fm$

by (*metis* (*no-types*, *lifting*) $\langle\forall m.\ \neg\ (m \in freevar\ (Forall\ var\ fm) \wedge \iota\ m \neq \vartheta\ m) \longrightarrow m \neq var \longrightarrow (if\ m \neq var\ then\ value\text{-}tm\ (\lambda k.\ if\ k = substuS\ var\ fm\ \vartheta\ then\ a\ else\ w\ k)\ (Cw, Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m)\ else\ a) \neq (if\ m \neq var\ then\ value\text{-}tm\ w\ (Cw, Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m)\ else\ a) \longrightarrow m = substuS\ var\ fm\ \vartheta\rangle$ $\langle\forall m.\ m \in freevar\ (Forall\ var\ fm) \wedge \iota\ m \neq \vartheta\ m \longrightarrow value\text{-}tm\ (\lambda k.\ if\ k = substuS\ var\ fm\ \vartheta\ then\ a\ else\ w\ k)\ (Cw, Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m) = value\text{-}tm\ w\ (Cw, Fw)\ (subst\vartheta S\ var\ fm\ \vartheta\ m)\rangle$)
from *P1 P2* **have**
$value\text{-}fm\ ?S\ ?w1\ (D, Cw, Fw, Rw)\ fm = value\text{-}fm\ ?S\ ?w2\ (D, Cw, Fw, Rw)\ fm$ **by** *auto*

from *this Calc1* **show**
value-fm ?S (λ *m. value-tm ?ws* (*Cw*, *Fw*)
(*subst*ϑ*S var fm* ϑ *m*)) (*D*, *Cw*, *Fw*, *Rw*) *fm* =
value-fm ?S (λ *m.* (*if m* $\neq$ *var then value-tm*
w (*Cw*, *Fw*) (*subst*ϑ*S var fm* ϑ *m*) *else a*))
(*D*, *Cw*, *Fw*, *Rw*) *fm* **by** *simp*
qed

have *Helper-right*: $\bigwedge$ *a. value-fm ?S* (λ*k. if k* = *var then a*
else (λ*m. value-tm w* (*Cw*, *Fw*) (ϑ *m*)) *k*) (*D*, *Cw*, *Fw*, *Rw*) *fm*
= *value-fm ?S* (λ *m.* (*if m* $\neq$ *var then value-tm*
w (*Cw*, *Fw*) (*subst*ϑ*S var fm* ϑ *m*) *else a*))
(*D*, *Cw*, *Fw*, *Rw*) *fm*
by (*smt DiffI freevar.simps*(*6*) *singletonD subst*ϑ*S.elims value-fm-locdetS*)

have *value-fm ?S* (*updt-w-subst* (*Cw*, *Fw*) *w* ϑ) (*D*, *Cw*, *Fw*, *Rw*) (*Forall var fm*)
= *value-fm ?S* (λ*m. value-tm w* (*Cw*, *Fw*) (ϑ *m*)) (*D*, *Cw*, *Fw*, *Rw*) (*Forall var fm*) **by** *simp*
moreover have ... = *myUni D* (λ *v. value-fm ?S* (λ*k. if k* = *var then v*
else (λ*m. value-tm w* (*Cw*, *Fw*) (ϑ *m*)) *k*) (*D*, *Cw*, *Fw*, *Rw*) *fm*) **by** *simp*
moreover have ... = *myUni D* (λ *v.*
value-fm ?S (λ *m.* (*if m* $\neq$ *var then value-tm*
w (*Cw*, *Fw*) (*subst*ϑ*S var fm* ϑ *m*) *else v*))
(*D*, *Cw*, *Fw*, *Rw*) *fm*) **using** *Helper-right* **by** *simp*
ultimately have *IdR*: *value-fm ?S* (*updt-w-subst* (*Cw*, *Fw*) *w* ϑ) (*D*, *Cw*, *Fw*, *Rw*) (*Forall var fm*)
= *myUni D* (λ *v. value-fm ?S* (λ *m.* (*if m* $\neq$ *var then value-tm*
w (*Cw*, *Fw*) (*subst*ϑ*S var fm* ϑ *m*) *else v*)) (*D*, *Cw*, *Fw*, *Rw*) *fm*) **by** *simp*

from *IdL IdR IdM* **show** *value-fm ?S w* (*D*, *Cw*, *Fw*, *Rw*) (*subst-formS* (*Forall var fm*) ϑ) =
value-fm ?S (*updt-w-subst* (*Cw*, *Fw*) *w* ϑ) (*D*, *Cw*, *Fw*, *Rw*) (*Forall var fm*) **by** *simp*
qed

fun *subst-form* :: ‹('*a*, '*b*, '*c*) *fm* $\Rightarrow$ *nat* $\Rightarrow$ ('*a*, '*b*) *tm* $\Rightarrow$ ('*a*, '*b*, '*c*) *fm*› **where**
subst-form fm n tm = *subst-formS fm* (ι (*n* := *tm*))

lemma *subst-form-xx-is-id*:
‹*subst-form* (*Forall x phi*) *x tm* = *Forall x phi*›
proof−
have *subst-formS* (*Forall x phi*) (ι (*x* := *tm*)) =
Forall (*substuS x phi* (ι (*x* := *tm*)))
(*subst-formS phi* (*subst*ϑ*S x phi* (ι (*x* := *tm*)))) **by** *simp*
then have *subst-formS* (*Forall x phi*) (ι (*x* := *tm*)) =
Forall x (*subst-formS phi* (*subst*ϑ*S x phi* (ι (*x* := *tm*)))) **by**(*simp add*:*substuSxx-is-id*)
then have *subst-formS* (*Forall x phi*) (ι (*x* := *tm*)) =

Forall x (*subst-formS phi* ι) **using** *substϑSxx-is-id* **by** *metis*
then show *subst-form* (*Forall x phi*) *x tm* =
Forall x phi **using** *subst-formS-ι-is-id* **by** *simp*
qed

lemma *subst-form-is-id*:
$x \notin$ *contvar fm* $\Longrightarrow$ (*subst-form fm x t*) = *fm*
proof(*induction fm arbitrary*: *x*)
case (*Rel Rs tl*)
from ‹$x \notin$ *contvar* (*Rel Rs tl*)› *freevar-contvar* **have** ‹$x \notin$ *freevar* (*Rel Rs tl*)› **by** *auto*
from ‹$x \notin$ *freevar* (*Rel Rs tl*)› **have** ‹$\forall$ *tm* $\in$ *set tl*. $x \notin$ *freevar-tm tm*› **by** *simp*
then have ‹$\forall$ *tm* $\in$ *set tl*. *subst-term tm x t* = *tm*› **using** *subst-term-is-id2* **by** *blast*
then show *?case* **by** (*simp add*: *map-idI subst-termS-test*)
next
case (*Equ tm1 tm2*)
from *this* **have** *A*: ‹$x \notin$ *freevar-tm tm1*› **and** *B*: ‹$x \notin$ *freevar-tm tm2*› **by** *auto*
from *A* **have** *1*: ‹*subst-term tm1 x t* = *tm1*› **using** *subst-term-is-id2* **by** *blast*
from *B* **have** *2*: ‹*subst-term tm2 x t* = *tm2*› **using** *subst-term-is-id2* **by** *blast*
from *1 2* **show** *?case* **by** (*simp add*: *subst-termS-test*)
next
case *Fal*
then show *?case* **by** *simp*
next
case (*And fm1 fm2*)
then show *?case* **by** *simp*
next
case (*Neg fm*)
then show *?case* **by** *simp*
next
case (*Forall var fm*)
from *this* **have** *H*: $x \notin$ *contvar* (*Forall var fm*) **by** *auto*
show *?case* **proof**(*cases x* = *var*)
case *True*
then show *?thesis* **using** *subst-form-xx-is-id* **by** *auto*
next
case *False*
have *subst-form* (*Forall var fm*) *x t*
= *subst-formS* (*Forall var fm*) (ι (*x* := *t*)) **by** *simp*
have *up*: *substuS var fm* (ι (*x* := *t*)) = *var*
using *H freevar-contvar* **by** *auto*
then have *ϑp*: *substϑS var fm* (ι (*x* := *t*)) = ι **using** *H freevar-contvar* **by** *fastforce*
show *?thesis* **using** *Forall.IH up ϑp subst-formS-ι-is-id* **by** *auto*
qed
qed

3.2 Further small definitions

fun *is-true-of* :: ‹(′a, ′b, ′c) fm ⇒ ′v list
⇒ (′v, ′a,′b,′c, ′mybool ::order) model ⇒ (′v, ′mybool) scheme ⇒ bool› **where**
is-true-of A val-list (*D*, *Cw*, *Fw*, *Rw*) (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*)
= (*myTrue* ≤ *value-fm′* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *val-list* (*D*, *Cw*, *Fw*, *Rw*) *A*)

fun *is-false-of* :: ‹(′a, ′b, ′c) fm ⇒ ′v list
⇒ (′v, ′a,′b,′c, ′mybool ::order) model ⇒ (′v, ′mybool) scheme ⇒ bool› **where**
is-false-of A val-list (*D*, *Cw*, *Fw*, *Rw*) (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*)
= (*myFalse* ≤ *value-fm′* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *val-list* (*D*, *Cw*, *Fw*, *Rw*) *A*)

lemma *theorem2A12a*: ⟦ *length* (*freevarL B*) = *length* (*dbar* :: ′*v list*)⟧
⟹ *is-true-of A dbar* (*D*, *Cw*, *Fw*, *Rw*) τ ∨ *is-false-of A dbar* (*D*, *Cw*, *Fw*, *Rw*) τ
by (*metis* (*full-types*) *bool2.exhaust is-false-of.simps is-true-of.simps leq2.simps*(*1*) *leq2.simps*(*2*) *less-eq-bool2-def*)

lemma *theorem2A12b*: ⟦ *length* (*freevarL B*) = *length* (*dbar* :: ′*v list*)⟧
⟹ ¬ (*is-true-of A dbar* (*D*, *Cw*, *Fw*, *Rw*) τ ∧ *is-false-of A dbar* (*D*, *Cw*, *Fw*, *Rw*) τ)
by (*metis* (*full-types*) *bool2.exhaust is-false-of.simps is-true-of.simps leq2.simps*(*3*) *leq2.simps*(*4*) *less-eq-bool2-def*)

lemma *theorem2A12c*: ⟦ *length* (*freevarL B*) = *length* (*dbar* :: ′*v list*)⟧
⟹ ¬ (*is-true-of A dbar* (*D*, *Cw*, *Fw*, *Rw*) κ ∧ *is-false-of A dbar* (*D*, *Cw*, *Fw*, *Rw*) κ)
by (*metis* (*full-types*) *bool3.exhaust is-false-of.simps is-true-of.simps leq3.simps*(*6*) *leq3.simps*(*7*) *leq3.simps*(*9*) *less-eq-bool3-def*)

lemma *theorem2A12d*: ⟦ *length* (*freevarL B*) = *length* (*dbar* :: ′*v list*)⟧
⟹ ¬ (*is-true-of A dbar* (*D*, *Cw*, *Fw*, *Rw*) μ ∧ *is-false-of A dbar* (*D*, *Cw*, *Fw*, *Rw*) μ)
by (*metis* (*full-types*) *bool3.exhaust is-false-of.simps is-true-of.simps leq3.simps*(*6*) *leq3.simps*(*7*) *leq3.simps*(*9*) *less-eq-bool3-def*)

lemma *theorem2A13a*: ⟦ *freevarL A* = []; *freevarL B* = []⟧
⟹ *is-true-of* (*And A B*) [] (*D*, *Cw*, *Fw*, *Rw*) τ ⟷
(*is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) τ ∧ *is-true-of B* [] (*D*, *Cw*, *Fw*, *Rw*) τ)
proof−
  assume *HA*: *freevarL A* = []
  assume *HB*: *freevarL B* = []
  show *is-true-of* (*And A B*) [] (*D*, *Cw*, *Fw*, *Rw*) τ ⟷
(*is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) τ ∧ *is-true-of B* [] (*D*, *Cw*, *Fw*, *Rw*) τ)
  proof
    assume *And-true*: *is-true-of* (*And A B*) [] (*D*, *Cw*, *Fw*, *Rw*) τ
    from *HA HB* **have** *HAB*: *freevarL* (*And A B*) = [] **by** *simp*
    from *And-true* **have** *And-true2*: *value-fm′* τ [] (*D*, *Cw*, *Fw*, *Rw*) (*And A B*)

= *t2*
 by (*metis* (*full-types*) *bool2.exhaust is-true-of.simps leq2.simps*(*4*) *less-eq-bool2-def*)
 from *HA HAB And-true2* **have** *ResA*: *value-fm′* τ [] (*D*, *Cw*, *Fw*, *Rw*) *A* = *t2*
 using τ-*and.elims* **by** *auto*
 from *HB HAB And-true2* **have** *ResB*: *value-fm′* τ [] (*D*, *Cw*, *Fw*, *Rw*) *B* = *t2*
 using τ-*and.elims* **by** *auto*
 from *ResA ResB* **show** (*is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) τ $\wedge$ *is-true-of B* []
(*D*, *Cw*, *Fw*, *Rw*) τ)
 by *simp*
 next
 assume *is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) τ $\wedge$ *is-true-of B* [] (*D*, *Cw*, *Fw*, *Rw*)
τ
 then have *HA*: *is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) τ **and** *HB*: *is-true-of B* []
(*D*, *Cw*, *Fw*, *Rw*) τ **by** *auto*
 from *HA* **have** *A-true*: *value-fm′* τ [] (*D*, *Cw*, *Fw*, *Rw*) *A* = *t2*
 by (*metis* (*full-types*) *bool2.exhaust is-true-of.simps leq2.simps*(*4*) *less-eq-bool2-def*)
 from *HB* **have** *B-true*: *value-fm′* τ [] (*D*, *Cw*, *Fw*, *Rw*) *B* = *t2*
 by (*metis* (*full-types*) *bool2.exhaust is-true-of.simps leq2.simps*(*4*) *less-eq-bool2-def*)
 from *A-true B-true* **have** *value-fm′* τ [] (*D*, *Cw*, *Fw*, *Rw*) (*And A B*) = *t2* **by**
auto
 then show *is-true-of* (*And A B*) [] (*D*, *Cw*, *Fw*, *Rw*) τ **by** *simp*
 qed
qed

lemma *theorem2A13b*: ⟦ *freevarL A* = []; *freevarL B* = []⟧
$\Longrightarrow$ *is-true-of* (*And A B*) [] (*D*, *Cw*, *Fw*, *Rw*) κ $\longleftrightarrow$
(*is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) κ $\wedge$ *is-true-of B* [] (*D*, *Cw*, *Fw*, *Rw*) κ)
proof−
 assume *HA*: *freevarL A* = []
 assume *HB*: *freevarL B* = []
 show *is-true-of* (*And A B*) [] (*D*, *Cw*, *Fw*, *Rw*) κ $\longleftrightarrow$
(*is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) κ $\wedge$ *is-true-of B* [] (*D*, *Cw*, *Fw*, *Rw*) κ)
 proof
 assume *And-true*: *is-true-of* (*And A B*) [] (*D*, *Cw*, *Fw*, *Rw*) κ
 from *HA HB* **have** *HAB*: *freevarL* (*And A B*) = [] **by** *simp*
 from *And-true* **have** *And-true2*: *value-fm′* κ [] (*D*, *Cw*, *Fw*, *Rw*) (*And A B*)
= *t3*
 by (*metis* (*full-types*) *bool3.exhaust is-true-of.simps leq3.simps*(*6*) *leq3.simps*(*7*)
less-eq-bool3-def)
 from *HA HAB And-true2* **have** *ResA*: *value-fm′* κ [] (*D*, *Cw*, *Fw*, *Rw*) *A* = *t3*
 using κ-*and.elims* **by** *auto*
 from *HB HAB And-true2* **have** *ResB*: *value-fm′* κ [] (*D*, *Cw*, *Fw*, *Rw*) *B* =
t3
 using κ-*and.elims* **by** *auto*
 from *ResA ResB* **show** (*is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) κ $\wedge$ *is-true-of B* []
(*D*, *Cw*, *Fw*, *Rw*) κ)
 by *simp*
 next
 assume *is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) κ $\wedge$ *is-true-of B* [] (*D*, *Cw*, *Fw*, *Rw*)

κ

then have *HA*: *is-true-of A* [] (*D*, *Cw*, *Fw*, *Rw*) *κ* **and** *HB*: *is-true-of B* [] (*D*, *Cw*, *Fw*, *Rw*) *κ* **by** *auto*

from *HA* **have** *A-true*: *value-fm′ κ* [] (*D*, *Cw*, *Fw*, *Rw*) *A* = *t3*

by (*metis* (*full-types*) *bool3.exhaust is-true-of.simps leq3.simps*(*6*) *leq3.simps*(*7*) *less-eq-bool3-def*)

from *HB* **have** *B-true*: *value-fm′ κ* [] (*D*, *Cw*, *Fw*, *Rw*) *B* = *t3*

by (*metis* (*full-types*) *bool3.exhaust is-true-of.simps leq3.simps*(*6*) *leq3.simps*(*7*) *less-eq-bool3-def*)

from *A-true B-true* **have** *value-fm′ κ* [] (*D*, *Cw*, *Fw*, *Rw*) (*And A B*) = *t3* **by** *auto*

then show *is-true-of* (*And A B*) [] (*D*, *Cw*, *Fw*, *Rw*) *κ* **by** *simp*

qed

qed

4 Definability of Truth

abbreviation *sentences* **where**
sentences ≡ { *fm*. *freevar fm* = {}}

fun *ground-mod* :: ‹*′mybool* ⇒ *′mybool* ⇒ (*′v*, *′a*, *′b*, *′c*, *′mybool*) *model* ⇒ *′c* ⇒ ((*′a*, *′b*, *′c*) *fm* ⇒ *′v*) ⇒ *bool*› **where**
ground-mod myFalse myTrue (*D*, *Cw*, *Fw*, *Rw*) *G c* =
((*inj c* ∧ *c‘ sentences* ⊆ *D*) ∧
(∀ *rsymb val-list*. *rsymb* ≠ *G*⟶
Rw rsymb val-list ∈ {*myTrue*, *myFalse*}))

fun *modplus* :: ‹(*′v*, *′a*, *′b*, *′c*, *′mybool*) *model* ⇒ *′c* ⇒ (*′v* ⇒ *′mybool*) ⇒ (*′v*, *′a*, *′b*, *′c*, *′mybool*) *model*›
where
modplus (*D*, *Cw*, *Fw*, *Rw*) *G g* = (*D*, *Cw*, *Fw*, *Rw* (*G* := (*λ val-list*. *g* (*hd val-list*))))

fun *jump* :: ‹(*′v*, *′mybool*) *scheme* ⇒ (*′v*, *′a*, *′b*, *′c*, *′mybool*) *model*
⇒ *′c* ⇒ ((*′a*, *′b*, *′c*) *fm* ⇒ *′v*) ⇒ (*′v* ⇒ *′mybool*) ⇒ (*′v* ⇒ *′mybool*)›
where
jump (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) (*D*, *Cw*, *Fw*, *Rw*) *G c g* =
(*if* (*ground-mod myFalse myTrue* (*D*, *Cw*, *Fw*, *Rw*) *G c*) *then*
(*λ A*. *if* (*A* ∈ *c‘ sentences*) *then*
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) (*λ x*. *undefined*)
(*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) (*inv c A*)
else myFalse) *else undefined*)

lemma *Fact*: ‹⟦ *ground-mod myFalse myTrue* (*D*, *Cw*, *Fw*, *Rw*) *G c*;
∀ *d*. *d* ∉ *c‘ sentences* ⟶ *g d* = *myFalse* ⟧
⟹ (∀ *d*. *jump* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) (*D*, *Cw*, *Fw*, *Rw*) *G c g d* = *g d*)
⟷ (∀ *A* ∈ *sentences*.
value-fm′ (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) [] (*modplus* (*D*, *Cw*, *Fw*,

Rw) *G g*) *A*
= *value-fm′* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) [*c A*] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) (*Rel G* [*Var 1*]))›
proof−
let *?S* = (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*)

assume *GM*: *ground-mod myFalse myTrue* (*D*, *Cw*, *Fw*, *Rw*) *G c*
then have *H2*: ‹*c*‘ *sentences* ⊆ *D*› **by** *simp*
assume *H1*: ‹∀ *d*. *d* ∉ *c*‘ *sentences* ⟶ *g d* = *myFalse*›

from *value-fm-locdet*
have *Locdet-fact*:
⋀ *s1 s2 A*. *A* ∈ *sentences* ⟶ *value-fm ?S s1* (*D*, *Cw*, *Fw*, *Rw*) *A* = *value-fm ?S s1* (*D*, *Cw*, *Fw*, *Rw*) *A* **by** *simp*
have *InD-fact*: ∀ *A* ∈ *sentences*. *c A* ∈ *D* **using** *H2* **by** *auto*

from *Locdet-fact InD-fact* **have** *Id-helper*:
∀ *A* ∈ *sentences*. *jump ?S* (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*)
= *value-fm ?S* (λ *x*. *undefined*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A*
using *GM* **by** *simp*

have ∀ *A* ∈ *sentences*. *freevarL A* = [] **using** *freevar-id* **by** (*smt mem-Collect-eq set-empty*)
then have ∀ *A* ∈ *sentences*.
value-fm′ ?S [] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A*
= *value-fm ?S* (*manufactured-assignment* [] [])
(*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* **by** *auto*
then have ∀ *A* ∈ *sentences*.
value-fm′ ?S [] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A*
= *value-fm ?S* (λ *x*. *undefined*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* **using** *value-fm-locdet* **by** *simp*
then have *Calculation1*: ∀ *A* ∈ *sentences*.
value-fm′ ?S [] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A*
= *jump ?S* (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*) **using** *Id-helper* **by** *auto*

have *Calculation2*: ∀ *A* ∈ *sentences*.
g (*c A*) = *value-fm′ ?S* [*c A*] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) (*Rel G* [*Var 1*]) **by** *simp*

show ‹ (∀ *d*. *jump* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) (*D*, *Cw*, *Fw*, *Rw*) *G c g d* = *g d*)
⟷ (∀ *A* ∈ *sentences*.
value-fm′ (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) [] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A*
= *value-fm′* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) [*c A*] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) (*Rel G* [*Var 1*]))› **proof**
assume ∀ *d*. *jump* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) (*D*, *Cw*, *Fw*, *Rw*) *G c g d* = *g d*
then have ∀ *A* ∈ *sentences*. *jump ?S* (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*) = *g* (*c A*)

using *InD-fact* **by** *blast*
hence $\forall$ *A* $\in$ *sentences*.
value-fm′ ?S [] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* = *g* (*c A*) **using** *Calculation1* **by** *simp*
thus $\forall$ *A* $\in$ *sentences*.
value-fm′ ?S [] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* = *value-fm′ ?S* [*c A*] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) (*Rel G* [*Var 1*])
using *Calculation2* **by** *metis*
next
assume ($\forall$ *A* $\in$ *sentences*.
value-fm′ (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) [] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A*
= *value-fm′* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) [*c A*] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) (*Rel G* [*Var 1*]))
hence $\forall$ *A* $\in$ *sentences*.
value-fm′ ?S [] (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* = *g* (*c A*) **using** *Calculation2* **by** *metis*
hence $\forall$ *A* $\in$ *sentences*. *jump ?S* (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*) = *g* (*c A*) **using** *Calculation1* **by** *simp*
then have *1*: $\forall$ *d*$\in$*D*. *jump ?S* (*D*, *Cw*, *Fw*, *Rw*) *G c g d* = *g d* **using** *H1 GM* **by** *auto*
have *2*: $\forall$ *d*. *d*$\notin$ *D* $\longrightarrow$ *jump ?S* (*D*, *Cw*, *Fw*, *Rw*) *G c g d* = *myFalse* **using** *GM* **by** *auto*
have $\forall$ *d*. *d*$\notin$ *D* $\longrightarrow$ *d* $\notin$ *c* ‘ *sentences* **using** *H2* **by** *auto*
then have *3*: $\forall$ *d*. *d*$\notin$*D* $\longrightarrow$ *g d* = *myFalse* **using** *H1* **by** *simp*
show $\forall$ *d*. *jump ?S* (*D*, *Cw*, *Fw*, *Rw*) *G c g d* = *g d* **using** *1 2 3*
by *blast*
qed
qed

lemma *jump-μ-monot*: ‹ *ground-mod f3 t3* (*D*, *Cw*, *Fw*, *Rw*) *G c* $\Longrightarrow$ *f* $\leq$ *g*
$\Longrightarrow$ *jump μ* (*D*, *Cw*, *Fw*, *Rw*) *G c f* $\leq$ *jump μ* (*D*, *Cw*, *Fw*, *Rw*) *G c g*›
proof−
assume *GM*: *ground-mod f3 t3* (*D*, *Cw*, *Fw*, *Rw*) *G c*
then have *S*: *c*‘ *sentences* $\subseteq$ *D* **by** *simp*
assume *H*: *f* $\leq$ *g*
hence *Rw* (*G* := (λ *val-list*. *f* (*hd val-list*)))
$\leq$ *Rw* (*G* := (λ *val-list*. *g* (*hd val-list*)))
by (*simp add*: *le-funD le-funI*)

then have *leqMod* (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G f*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*)
by *simp*
from *this monot-in-μ* **have** *Res*: $\bigwedge$ *s A*.
value-fm μ s (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G f*) *A*
$\leq$ *value-fm μ s* (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* **by** *fastforce*

have $\forall$ *A* $\in$ *sentences*.
jump μ (*D*, *Cw*, *Fw*, *Rw*) *G c f* (*c A*) $\leq$ *jump μ* (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c*

A) **proof**
fix *A*::(*'c*, *'b*, *'d*) *fm*
assume *Asent*: ‹*A* ∈ *sentences*›

from *S this* **have** *A*: *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c f* (*c A*) =
value-fm μ (λ *x*. *undefined*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G f*) *A* **using** *GM* **by** *simp*

from *S Asent* **have** *B*: *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*) =
value-fm μ (λ *x*. *undefined*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* **using** *GM* **by** *simp*

show *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c f* (*c A*) ≤ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*)
using *A B Res* **by** *simp*
qed
from *this* **have** *Res1*: ∀ *d* ∈ *c*‘ *sentences*.
jump μ (*D*, *Cw*, *Fw*, *Rw*) *G c f d* ≤ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g d* **using** *GM* **by** *simp*

have *A*: ∀ *d*. *d* ∉ *c*‘ *sentences* ⟶ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c f d* = *f3* **using** *GM* **by** *simp*
have *B*: ∀ *d*. *d* ∉ *c*‘ *sentences* ⟶ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g d* = *f3* **using** *GM* **by** *simp*
from *A B* **have** *Res2*: ∀ *d*. *d* ∉ *c*‘ *sentences* ⟶ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c f d* ≤ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g d* **by** *simp*

have ∀ *d*. *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c f d* ≤ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g d*
using *Res1 Res2* **by** *smt*
then show *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c f* ≤ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g*
using *le-funI* **by** *blast*
qed

lemma *jump-κ-monot*: ‹ *ground-mod f3 t3* (*D*, *Cw*, *Fw*, *Rw*) *G c* ⟹ *f* ≤ *g*
⟹ *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c f* ≤ *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c g*›
proof−
assume *GM*: *ground-mod f3 t3* (*D*, *Cw*, *Fw*, *Rw*) *G c*
then have *S*: *c*‘ *sentences* ⊆ *D* **by** *simp*
assume *H*: *f* ≤ *g*
hence *Rw* (*G* := (λ *val-list*. *f* (*hd val-list*)))
≤ *Rw* (*G* := (λ *val-list*. *g* (*hd val-list*)))
by (*simp add*: *le-funD le-funI*)

then have *leqMod* (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G f*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*)
by *simp*
from *this monot-in-κ* **have** *Res*: ⋀ *s A*.
value-fm κ *s* (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G f*) *A*

$\leq$ *value-fm* κ *s* (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* **by** *fastforce*

have $\forall$ *A* $\in$ *sentences*.
jump κ (*D*, *Cw*, *Fw*, *Rw*) *G c f* (*c A*) $\leq$ *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*) **proof**
fix *A*::('*c*, '*b*, '*d*) *fm*
assume *Asent*: ‹*A* $\in$ *sentences*›

from *S this* **have** *A*: *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c f* (*c A*) =
value-fm κ (λ *x*. *undefined*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G f*) *A* **using** *GM* **by** *simp*

from *S Asent* **have** *B*: *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*) =
value-fm κ (λ *x*. *undefined*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* **using** *GM* **by** *simp*

show *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c f* (*c A*) $\leq$ *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*)
using *A B Res* **by** *simp*
qed
from *this* **have** *Res1*: $\forall$ *d* $\in$ *c*‘ *sentences*.
jump κ (*D*, *Cw*, *Fw*, *Rw*) *G c f d* $\leq$ *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c g d* **using** *GM* **by** *simp*
have *Res2*: $\forall$ *d*. *d* $\notin$ *c*‘ *sentences* $\longrightarrow$ *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c f d* $\leq$ *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c g d* **by** *auto*

have $\forall$ *d*. *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c f d* $\leq$ *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c g d*
using *Res1 Res2* **by** *smt*
then show *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c f* $\leq$ *jump* κ (*D*, *Cw*, *Fw*, *Rw*) *G c g*
using *le-funI* **by** *blast*
qed

lemma *jump-ν-monot*: ‹ *ground-mod f4 t4* (*D*, *Cw*, *Fw*, *Rw*) *G c* $\Longrightarrow$ *f* $\leq$ *g*
$\Longrightarrow$ *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c f* $\leq$ *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g*›
proof−
assume *GM*: *ground-mod f4 t4* (*D*, *Cw*, *Fw*, *Rw*) *G c*
then have *S*: *c*‘ *sentences* $\subseteq$ *D* **by** *simp*
assume *H*: *f* $\leq$ *g*
hence *Rw* (*G* := (λ *val-list*. *f* (*hd val-list*)))
$\leq$ *Rw* (*G* := (λ *val-list*. *g* (*hd val-list*)))
by (*simp add*: *le-funD le-funI*)

then have *leqMod* (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G f*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*)
by *simp*
from *this monot-in-ν* **have** *Res*: $\bigwedge$ *s A*.
value-fm ν *s* (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G f*) *A*
$\leq$ *value-fm* ν *s* (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* **by** *fastforce*

have ∀ $A \in$ *sentences*.
jump ν (*D*, *Cw*, *Fw*, *Rw*) *G c f* (*c A*) $\leq$ *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*) **proof**
fix *A*::('*c*, '*b*, '*d*) *fm*
assume *Asent*: ‹*A* ∈ *sentences*›

from *S this* **have** *A*: *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c f* (*c A*) =
value-fm ν (λ *x*. *undefined*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G f*) *A* **using** *GM* **by** *simp*

from *S Asent* **have** *B*: *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*) =
value-fm ν (λ *x*. *undefined*) (*modplus* (*D*, *Cw*, *Fw*, *Rw*) *G g*) *A* **using** *GM* **by** *simp*

show *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c f* (*c A*) $\leq$ *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g* (*c A*)
using *A B Res* **by** *simp*
qed
from *this* **have** *Res1*: ∀ *d* ∈ *c*‘ *sentences*.
jump ν (*D*, *Cw*, *Fw*, *Rw*) *G c f d* $\leq$ *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g d* **using** *GM* **by** *simp*
have *Res2*: ∀ *d*. *d* ∉ *c*‘ *sentences* ⟶ *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c f d* $\leq$ *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g d* **by** *auto*

have ∀ *d*. *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c f d* $\leq$ *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g d*
using *Res1 Res2* **by** *smt*
then show *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c f* $\leq$ *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g*
using *le-funI* **by** *blast*
qed

lemma *μ-fixed-point-prop*:
‹*ground-mod f3 t3* (*D*, *Cw*, *Fw*, *Rw*) *G c* ⟹
(∃ *g*. *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g* = *g*)›
proof−
assume *GM*: *ground-mod f3 t3* (*D*, *Cw*, *Fw*, *Rw*) *G c*
then have *H*: *c*‘ *sentences* ⊆ *D* **by** *simp*
let *?U* = (*UNIV* :: ('*v* ⇒ *bool3*) *set*)
have *1*: *ccpo ?U*
using *function-space-ccpo-bool3* **by** *simp*

have ∀ *g1 g2*.
g1 $\leq$ *g2* ⟶ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g1*
$\leq$ *jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c g2*
using *jump-μ-monot GM* **by** *metis*
then have *2*: *monot* (*jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c*) **by** *simp*
from *1 2 VisserFixp* **have** *ccpo* (*FixPs ?U* (*jump* μ (*D*, *Cw*, *Fw*, *Rw*) *G c*))
by *blast*

from *this ccpo-least-element* **have** $\exists g.\ g \in (FixPs\ ?U\ (jump\ \mu\ (D,\ Cw,\ Fw,\ Rw)\ G\ c))$ **by** *auto*
then obtain *g* **where** *Hg*: ‹$g \in (FixPs\ ?U\ (jump\ \mu\ (D,\ Cw,\ Fw,\ Rw)\ G\ c))$› **by** *auto*
show $\exists\ g.\ jump\ \mu\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g = g$ **proof**
show $jump\ \mu\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g = g$ **using** *FixPs-def Hg* **by** *blast*
qed
qed

lemma *κ-fixed-point-prop*:
‹$ground\text{-}mod\ f3\ t3\ (D,\ Cw,\ Fw,\ Rw)\ G\ c \Longrightarrow$
$(\exists\ g.\ jump\ \kappa\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g = g\ )$›
proof−
assume *GM*: $ground\text{-}mod\ f3\ t3\ (D,\ Cw,\ Fw,\ Rw)\ G\ c$
then have *H*: $c\text{`}\ sentences \subseteq D$ **by** *simp*
let $?U = (\ UNIV :: ('v \Rightarrow bool3)\ set)$
have *1*: *ccpo ?U*
using *function-space-ccpo-bool3* **by** *simp*

have $\forall\ g1\ g2.$
$g1 \leq g2 \longrightarrow jump\ \kappa\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g1$
$\leq jump\ \kappa\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g2$
using *jump-κ-monot GM* **by** *metis*
then have *2*: $monot\ (jump\ \kappa\ (D,\ Cw,\ Fw,\ Rw)\ G\ c)$ **by** *simp*
from *1 2 VisserFixp* **have** $ccpo\ (\ FixPs\ ?U\ (jump\ \kappa\ (D,\ Cw,\ Fw,\ Rw)\ G\ c))$
by *blast*
from *this ccpo-least-element* **have** $\exists g.\ g \in (FixPs\ ?U\ (jump\ \kappa\ (D,\ Cw,\ Fw,\ Rw)\ G\ c))$ **by** *auto*
then obtain *g* **where** *Hg*: ‹$g \in (FixPs\ ?U\ (jump\ \kappa\ (D,\ Cw,\ Fw,\ Rw)\ G\ c))$› **by** *auto*
show $\exists\ g.\ jump\ \kappa\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g = g$ **proof**
show $jump\ \kappa\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g = g$ **using** *FixPs-def Hg* **by** *blast*
qed
qed

lemma *ν-fixed-point-prop*:
‹$ground\text{-}mod\ f4\ t4\ (D,\ Cw,\ Fw,\ Rw)\ G\ c \Longrightarrow$
$(\exists\ g.\ jump\ \nu\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g = g\ )$›
proof−
assume *GM*: $ground\text{-}mod\ f4\ t4\ (D,\ Cw,\ Fw,\ Rw)\ G\ c$
then have *H*: $c\text{`}\ sentences \subseteq D$ **by** *simp*
let $?U = (\ UNIV :: ('v \Rightarrow bool4)\ set)$
have *1*: *ccpo ?U*
using *function-space-ccpo-bool4* **by** *simp*

have $\forall\ g1\ g2.$
$g1 \leq g2 \longrightarrow jump\ \nu\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g1$
$\leq jump\ \nu\ (D,\ Cw,\ Fw,\ Rw)\ G\ c\ g2$
using *jump-ν-monot GM* **by** *metis*

then have *2*: *monot* (*jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c*) **by** *simp*
from *1 2 VisserFixp* **have** *ccpo* (*FixPs ?U* (*jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c*))
by *blast*
from *this ccpo-least-element* **have** $\exists$ *g*. *g* $\in$ (*FixPs ?U* (*jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c*)) **by** *auto*
then obtain *g* **where** *Hg*: ‹*g* $\in$ (*FixPs ?U* (*jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c*))› **by** *auto*
show $\exists$ *g*. *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g* = *g* **proof**
show *jump* ν (*D*, *Cw*, *Fw*, *Rw*) *G c g* = *g* **using** *FixPs-def Hg* **by** *blast*
qed
qed

5 The Transfer Theorem

datatype (*'N*) *fmP*
= *RelG* ‹*'N*›

| *FalP*
| *AndP* ‹*'N fmP*› ‹*'N fmP*›
| *NegP* ‹*'N fmP*›

fun *liftfmP* :: ‹*'c* ⇒ (*'N* ⇒ *'b*) ⇒ *'N fmP* ⇒ (*'a*, *'b*, *'c*) *fm*› **where**
liftfmP G namec (*RelG n*) = *Rel G* [*Const* (*namec n*)] |
liftfmP G namec FalP = *Fal* |
liftfmP G namec (*AndP fm1 fm2*) = *And* (*liftfmP G namec fm1*) (*liftfmP G namec fm2*) |
liftfmP G namec (*NegP fm*) = *Neg* (*liftfmP G namec fm*)

type-synonym (*'N*, *'mybool*) *Pmodel*
= ‹(*'N* ⇒ *'mybool*)›

type-synonym (*'mybool*) *Pscheme*
= ‹*'mybool* × *'mybool* × (*'mybool* ⇒ *'mybool*) × (*'mybool* ⇒ *'mybool* ⇒ *'mybool*)›

fun *value-fmP* :: ‹(*'mybool*) *Pscheme* ⇒ (*'N*, *'mybool*) *Pmodel* ⇒ *'N fmP* ⇒ *'mybool*› **where**
value-fmP (*myFalse*, *myTrue*, *myNot*, *myAnd*) *v FalP* = *myFalse* |
value-fmP (*myFalse*, *myTrue*, *myNot*, *myAnd*) *v* (*RelG a*) = *v a* |

value-fmP (*myFalse*, *myTrue*, *myNot*, *myAnd*) *v* (*AndP fm1 fm2*) = (*myAnd* (*value-fmP* (*myFalse*, *myTrue*, *myNot*, *myAnd*) *v fm1*) (*value-fmP* (*myFalse*, *myTrue*, *myNot*, *myAnd*) *v fm2*)) |
value-fmP (*myFalse*, *myTrue*, *myNot*, *myAnd*) *v* (*NegP f*) = (*myNot* (*value-fmP* (*myFalse*, *myTrue*, *myNot*, *myAnd*) *v f*))

fun *value-fmPc* **where**
(*value-fmPc* (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) *v fm*)
= (*value-fmP* (*myFalse*, *myTrue*, *myNot*, *myAnd*) *v fm*)

type-synonym $'N$ *reference-list*
 $= \langle 'N \Rightarrow 'N\ fmP\rangle$

datatype *toy-type* = *toy-typeA* | *toy-typeB*

fun *testR* ::(*toy-type reference-list*) **where**
testR toy-typeA = *NegP* (*RelG toy-typeA*) |
testR toy-typeB = *AndP* (*NegP* (*RelG toy-typeA*)) (*RelG toy-typeB*)

fun *toy-type-nn* **where** *toy-type-nn* (*x* :: *toy-type*) = *n3*
fun *toy-type-nt* **where** *toy-type-nt toy-typeA* = *n3* | *toy-type-nt toy-typeB* = *t3*
fun *toy-type-tn* **where** *toy-type-tn toy-typeA* = *t3* | *toy-type-tn toy-typeB* = *n3*
fun *toy-type-fn* **where** *toy-type-fn toy-typeA* = *f3* | *toy-type-fn toy-typeB* = *n3*
fun *toy-type-nf* **where** *toy-type-nf toy-typeA* = *n3* | *toy-type-nf toy-typeB* = *f3*
fun *toy-type-tt* **where** *toy-type-tt* (*x* :: *toy-type*) = *t3*
fun *toy-type-ff* **where** *toy-type-ff* (*x* :: *toy-type*) = *f3*
fun *toy-type-ft* **where** *toy-type-ft toy-typeA* = *f3* | *toy-type-ft toy-typeB* = *t3*
fun *toy-type-tf* **where** *toy-type-tf toy-typeA* = *t3* | *toy-type-tf toy-typeB* = *f3*

lemma ⟨*toy-type-nn* ≤ (*f* :: *toy-type* ⇒ *bool3*)⟩
 by (*metis* (*full-types*) *bool3.exhaust insertI1 insert-commute le-funI leq3.simps*(*1*)
leq3.simps(*4*) *less-eq-bool3-def order-example1B supRs-def toy-type-nn.elims*)
lemma ⟨*toy-type-nt* ≤ *toy-type-tt*⟩
 by (*metis* (*full-types*) *le-funI leq3.simps*(*4*) *less-eq-bool3-def order-refl toy-type.exhaust*
toy-type-nt.simps(*1*) *toy-type-nt.simps*(*2*) *toy-type-tt.elims*)

fun *jumpP* :: ⟨*'mybool Pscheme* ⇒ *'N reference-list* ⇒ (*'N*, *'mybool*) *Pmodel*
 ⇒ (*'N*, *'mybool*) *Pmodel*⟩
 where
jumpP (*myFalse*, *myTrue*, *myNot*, *myAnd*) *R v* =
 (λ *a*. *value-fmP* (*myFalse*, *myTrue*, *myNot*, *myAnd*) *v* (*R a*))

lemma ⟨*fixedp toy-type-nn* (*jumpP* (*f3*, *t3*, κ-*not*, κ-*and*) *testR*)⟩
proof−
 have ⟨∀ *a*. *jumpP* (*f3*, *t3*, κ-*not*, κ-*and*) *testR toy-type-nn a* = *toy-type-nn a*⟩
 proof fix *a* **show** ⟨*jumpP* (*f3*, *t3*, κ-*not*, κ-*and*) *testR toy-type-nn a* = *toy-type-nn*
a⟩
  by(*cases a*; *simp*) **qed**
 then have *jumpP* (*f3*, *t3*, κ-*not*, κ-*and*) *testR*
  toy-type-nn = *toy-type-nn* **by**(*simp add*: *ext*)
 then show *?thesis* **by** (*simp add*: *fixedp-def*)
qed

lemma ⟨*fixedp toy-type-nf* (*jumpP* (*f3*, *t3*, κ-*not*, κ-*and*) *testR*)⟩
proof−
 have ⟨∀ *a*. *jumpP* (*f3*, *t3*, κ-*not*, κ-*and*) *testR toy-type-nf a* = *toy-type-nf a*⟩
 proof fix *a* **show** ⟨*jumpP* (*f3*, *t3*, κ-*not*, κ-*and*) *testR toy-type-nf a* = *toy-type-nf*
a⟩
  by(*cases a*; *simp*) **qed**

then have *jumpP* (*f3*, *t3*, *κ-not*, *κ-and*) *testR*
toy-type-nf = *toy-type-nf* **by**(*simp add*: *ext*)
then show *?thesis* **by** (*simp add*: *fixedp-def*)
qed

fun *is-neutral-name* :: ‹(′*v* , ′*a*, ′*b*, ′*c*, ′*mybool*) *model* ⇒ ′*v set* ⇒ ′*b* ⇒ *bool*›
where
is-neutral-name (*D*, *Cw*, *Fw*, *Rw*) *X a* = ((*Cw a*) ∉ *X*)

fun *is-neutral-Rsymb* :: ‹(′*v* , ′*a*, ′*b*, ′*c*, ′*mybool*) *model* ⇒ ′*v set* ⇒ ′*c* ⇒ *bool*›
where
is-neutral-Rsymb (*D*, *Cw*, *Fw*, *Rw*) *X F* = (∀ *val-list val-list*′.
(∀ *x* ∈ *set val-list*. *x* ∈ *X*) ∧ (∀ *y* ∈ *set val-list*′. *y* ∈ *X*)
⟶ *Rw F val-list* = *Rw F val-list*′)

fun *is-neutral-Fsymb* :: ‹(′*v* , ′*a*, ′*b*, ′*c*, ′*mybool*) *model* ⇒ ′*v set* ⇒ ′*a* ⇒ *bool*›
where
is-neutral-Fsymb (*D*, *Cw*, *Fw*, *Rw*) *X f* = (∀ *val-list val-list*′.
(∀ *x* ∈ *set val-list*. *x* ∈ *X*) ∧ (∀ *y* ∈ *set val-list*′. *y* ∈ *X*)
⟶ *Fw f val-list* = *Fw f val-list*′)

fun *is-quant-enrichment* :: ‹′*mybool* ⇒ ′*mybool* ⇒ (′*v*, ′*a*, ′*b*, ′*c*, ′*mybool*) *model* ⇒ ′*c* ⇒ ((′*a*, ′*b*, ′*c*) *fm* ⇒ ′*v*) ⇒ (′*N* ⇒ ′*b*) ⇒ ((′*a*, ′*b*, ′*c*) *fm* ⇒ ′*b*) ⇒ ′*N reference-list* ⇒ *bool*›
where
is-quant-enrichment myFalse myTrue (*D*, *Cw*, *Fw*, *Rw*) *G c Pnamec quotnamec R* = (
(*ground-mod myFalse myTrue* (*D*, *Cw*, *Fw*, *Rw*) *G c*) ∧
(*inj Pnamec* ∧ *inj quotnamec*) ∧
(∀ *A* ∈ *sentences*. (*Cw* (*quotnamec A*)) = *c A*) ∧
(∀ *n*:: ′*N*. *Cw* (*Pnamec n*) = *c* (*liftfmP G Pnamec* (*R n*))) ∧
(∀ *b*:: ′*b*. (*b* ∉ *range Pnamec* ∧ *b* ∉ *quotnamec* ‘ *sentences*)
⟶ *is-neutral-name* (*D*, *Cw*, *Fw*, *Rw*) (*c*‘ *sentences*) *b*) ∧
(∀ *a*:: ′*a*. *is-neutral-Fsymb* (*D*, *Cw*, *Fw*, *Rw*) (*c*‘ *sentences*) *a*) ∧
(∀ *c*′:: ′*c*. *is-neutral-Rsymb* (*D*, *Cw*, *Fw*, *Rw*) (*c*‘ *sentences*) *c*′))

6 Generalisations

lemma (*n4*, *n4*) ≤ (*b4*, *b4*)
by (*simp add*: *less-eq-bool4-def*)

lemma *product-ccpo*: ‹⟦ *ccpo* (*A* :: (′*a* :: *order*) *set*);
ccpo (*B* :: (′*b* :: *order*) *set*) ⟧
⟹ *ccpo* (*A* × *B*)›
proof−
assume *HA*: *ccpo A*
assume *HB*: *ccpo B*
have ∀ *X*⊆ (*A* × *B*). *consi X* (*A* × *B*) ⟶ (∃ *b*∈ (*A*× *B*). *supRs b X* (*A*×*B*))
proof

fix X
show $X \subseteq A \times B \longrightarrow$ *consi* X $(A \times B) \longrightarrow (\exists\, b{\in}A \times B.$ *supRs* b X $(A \times B))$ **proof**
assume *HX*: $X \subseteq (A \times B)$
show *consi* X $(A{\times}B) \longrightarrow (\exists\, b{\in}\ (A{\times}\ B).$ *supRs* b X $(A{\times}B))$ **proof**
assume *HC*: *consi* X $(A{\times}B)$
let *?X1* = *fst* ‘ X
from *HX HC* **have** *?X1* $\subseteq A$ **by** *auto*
from *HC* **have** $\forall\, c \in X.\ \forall\ d \in X.\ \exists\, b \in (A{\times}B).\ c \leq b \wedge d \leq b$ **by** (*simp add*: *consi-def*)
from *this* **have** $\forall\, x{\in}$ *?X1*. $\forall\, y{\in}$*?X1*. $\exists\, b{\in}A.\ x \leq b \wedge y \leq b$ **by** *fastforce*
from *this* **have** *consi ?X1 A* **by**(*simp add*: *consi-def*)
from *this HA* **have** $\exists$ *ba* $\in A$. *supRs ba ?X1 A* **using** *ccpo-def*
using ‹*fst* ‘ $X \subseteq A$› **by** *blast*
then obtain *ba* **where** *ba* $\in A$ **and** *supRs ba ?X1 A* **by** *auto*

let *?X2* = *snd* ‘ X
from *HX HC* **have** *?X2* $\subseteq B$ **by** *auto*
from *HC* **have** $\forall\, c \in X.\ \forall\ d \in X.\ \exists\, b \in (A{\times}B).\ c \leq b \wedge d \leq b$ **by** (*simp add*: *consi-def*)
from *this* **have** $\forall\, x{\in}$ *?X2*. $\forall\, y{\in}$*?X2*. $\exists\, b{\in}B.\ x \leq b \wedge y \leq b$ **by** *fastforce*
from *this* **have** *consi ?X2 B* **by**(*simp add*: *consi-def*)
from *this HB* **have** $\exists$ *bb* $\in B$. *supRs bb ?X2 B* **using** *ccpo-def*
using ‹*snd* ‘ $X \subseteq B$› **by** *blast*
then obtain *bb* **where** *bb* $\in B$ **and** *supRs bb ?X2 B* **by** *auto*

show $(\exists\, b{\in}\ (A{\times}\ B).$ *supRs* b X $(A{\times}B))$ **proof**
have *1*: (*ba*, *bb*) $\in A \times B$ **using** ‹*ba* $\in A$› ‹*bb* $\in B$› **by** *auto*
have *2*: $(\forall\, y{\in}X.\ y \leq$ (*ba*, *bb*)) **using** *HX* ‹*supRs bb ?X2 B*› ‹*supRs ba ?X1 A*›
by (*simp add*: *less-eq-prod-def supRs-def*)
have *3*: $(\forall\, y{\in}A \times B.\ (\forall\, ya{\in}X.\ ya \leq y) \longrightarrow$ (*ba*, *bb*) $\leq y$) **using** ‹*supRs bb ?X2 B*› ‹*supRs ba ?X1 A*›
by (*simp add*: *less-eq-prod-def supRs-def*)
show *supRs* (*ba*,*bb*) X $(A \times B)$ **using** *1 2 3* **by**(*simp add*:*supRs-def*)
next
show (*ba*, *bb*) $\in A \times B$ **using** ‹*ba* $\in A$› ‹*bb* $\in B$› **by** *auto*
qed
qed
qed
qed
then show *ccpo* $(A \times B)$ **by**(*simp add*: *ccpo-def*)
qed

type-synonym (*'w*, *'v*, *'a*, *'b*, *'c*, *'mybool*) *Wmodel*
= ‹ *'v set* × (*'w* ⇒ (*'v*, *'a*,*'b*,*'c*) *const-mod*) × (*'w* ⇒ (*'v*, *'a*,*'b*,*'c*) *func-mod*)
× (*'w* ⇒ (*'v*, *'a*,*'b*,*'c*, *'mybool*) *rela-mod*)›

fun *ground-Wmod* :: ‹*'mybool* ⇒ *'mybool* ⇒ (*'w*, *'v*, *'a*, *'b*, *'c*, *'mybool*) *Wmodel*

$\Rightarrow$ *'c* $\Rightarrow$ ((*'a*, *'b*, *'c*) *fm* $\Rightarrow$ *'v*) $\Rightarrow$ *bool*› **where**
ground-Wmod myFalse myTrue (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c* =
((*inj c* $\wedge$ *c*' *sentences* $\subseteq$ *D*) $\wedge$
($\forall$ *w rsymb val-list*. *rsymb* $\neq$ *G* $\longrightarrow$
$\mathfrak{R}w$ *w rsymb val-list* $\in$ {*myTrue*, *myFalse*}))

function *jumpW* :: ‹(*'v*, *'mybool*) *scheme* $\Rightarrow$ (*'w*, *'v*, *'a*, *'b*, *'c*, *'mybool*) *Wmodel*
$\Rightarrow$ *'c* $\Rightarrow$ ((*'a*, *'b*, *'c*) *fm* $\Rightarrow$ *'v*) $\Rightarrow$ (*'w* $\Rightarrow$ *'v* $\Rightarrow$ *'mybool*) $\Rightarrow$ (*'w* $\Rightarrow$ *'v* $\Rightarrow$ *'mybool*)›
where
jumpW (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g* =
(*if* (*ground-Wmod myFalse myTrue* (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c*) *then*
(λ *w A*. *if* (*A* $\in$ *c*' *sentences*) *then*
value-fm (*myFalse*, *myTrue*, *myNot*, *myAnd*, *myUni*) (λ *x*. *undefined*)
(*D*, $\mathfrak{C}w$ *w*, $\mathfrak{F}w$ *w*, ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*g w*) (*hd val-list*)))) (*inv c A*)
else myFalse) *else undefined*)
apply *auto*[*1*]
by *blast*
termination by *lexicographic-order*

lemma *jumpW-μ-monot*: ‹ *ground-Wmod f3 t3* (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c* $\Longrightarrow$ *f* $\leq$ *g*
$\Longrightarrow$ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f* $\leq$ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g*›
proof−
assume *GM*: *ground-Wmod f3 t3* (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c*
then have *S*: *c*' *sentences* $\subseteq$ *D* **by** *simp*
assume *H*: *f* $\leq$ *g*
hence $\forall$ *w*. ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*f w*) (*hd val-list*)))
$\leq$ ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*g w*) (*hd val-list*)))
by (*simp add*: *H le-funD le-funI*)

then have $\forall$ *w*. *leqMod* (*D*, $\mathfrak{C}w$ *w*, $\mathfrak{F}w$ *w*, ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*f w*) (*hd val-list*))))
(*D*, $\mathfrak{C}w$ *w*, $\mathfrak{F}w$ *w*, ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*g w*) (*hd val-list*))))
by *simp*
from *this monot-in-μ* **have** *Res*: $\bigwedge$ *s A w*.
value-fm μ *s* (*D*, $\mathfrak{C}w$ *w*, $\mathfrak{F}w$ *w*, ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*f w*) (*hd val-list*))))
A
$\leq$ *value-fm* μ *s* (*D*, $\mathfrak{C}w$ *w*, $\mathfrak{F}w$ *w*, ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*g w*) (*hd val-list*))))
A **by** *fastforce*

have $\forall$ *w*. $\forall$ *A* $\in$ *sentences*.
jumpW μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w* (*c A*) $\leq$ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G*
c g w (*c A*) **proof**
fix *w*
show $\forall$ *A* $\in$ *sentences*.
jumpW μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w* (*c A*) $\leq$ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G*

c g w (*c A*) **proof**
fix *A*::(*'d*, *'c*, *'e*) *fm*
assume *Asent*: ‹*A* ∈ *sentences*›

from *S this* **have** *A*: *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w* (*c A*) =
value-fm μ (λ *x*. *undefined*) (*D*, $\mathfrak{C}w$ *w*, $\mathfrak{F}w$ *w*, ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*f w*) (*hd val-list*)))) *A* **using** *GM* **by** *simp*

from *S Asent* **have** *B*: *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g w* (*c A*) =
value-fm μ (λ *x*. *undefined*) (*D*, $\mathfrak{C}w$ *w*, $\mathfrak{F}w$ *w*, ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*g w*) (*hd val-list*)))) *A* **using** *GM* **by** *simp*

show *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w* (*c A*) ≤ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g w* (*c A*)
using *A B Res* **by** *simp*
qed
qed
from *this* **have** *Res1*: ∀ *w*. ∀ *d* ∈ *c*‘ *sentences*.
jumpW μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w d* ≤ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g w d* **using** *GM* **by** *simp*

have *A*: ∀ *w d*. *d* ∉ *c*‘ *sentences* ⟶ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w d* = *f3* **using** *GM* **by** *simp*
have *B*: ∀ *w d*. *d* ∉ *c*‘ *sentences* ⟶ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g w d* = *f3* **using** *GM* **by** *simp*
from *A B* **have** *Res2*: ∀ *w d*. *d* ∉ *c*‘ *sentences* ⟶ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w d* ≤ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g w d* **by** *simp*

have ∀ *w d*. *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w d* ≤ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g w d*
using *Res1 Res2* **by** *smt*
then show *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f* ≤ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g*
using *le-funI* **by** *smt*
qed

lemma *jumpW-*κ*-monot*: ‹ *ground-Wmod f3 t3* (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c* ⟹ *f* ≤ *g* ⟹ *jumpW* κ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f* ≤ *jumpW* κ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g*›
proof−
assume *GM*: *ground-Wmod f3 t3* (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c*
then have *S*: *c*‘ *sentences* ⊆ *D* **by** *simp*
assume *H*: *f* ≤ *g*
hence ∀ *w*. ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*f w*) (*hd val-list*)))
≤ ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*g w*) (*hd val-list*)))
by (*simp add*: *H le-funD le-funI*)

then have ∀ *w*. *leqMod* (*D*, $\mathfrak{C}w$ *w*, $\mathfrak{F}w$ *w*, ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*f w*) (*hd val-list*))))
(*D*, $\mathfrak{C}w$ *w*, $\mathfrak{F}w$ *w*, ($\mathfrak{R}w$ *w*) (*G* := (λ *val-list*. (*g w*) (*hd val-list*))))

by *simp*
from *this monot-in-κ* **have** *Res*: $\bigwedge$ *s A w*.
value-fm κ *s* (D, $\mathfrak{C}w$ w, $\mathfrak{F}w$ w, ($\mathfrak{R}w$ w) (G := (λ *val-list*. (f w) (*hd val-list*)))) A
$\leq$ *value-fm* κ *s* (D, $\mathfrak{C}w$ w, $\mathfrak{F}w$ w, ($\mathfrak{R}w$ w) (G := (λ *val-list*. (g w) (*hd val-list*)))) A **by** *fastforce*

have $\forall$ w. $\forall$ $A \in$ *sentences*.
jumpW κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c f w (c A) $\leq$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c g w (c A) **proof**
fix w
show $\forall$ $A \in$ *sentences*.
jumpW κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c f w (c A) $\leq$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c g w (c A) **proof**
fix A::($'d$, $'c$, $'e$) *fm*
assume *Asent*: ‹$A \in$ *sentences*›

from *S this* **have** A: *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c f w (c A) =
value-fm κ (λ x. *undefined*) (D, $\mathfrak{C}w$ w, $\mathfrak{F}w$ w, ($\mathfrak{R}w$ w) (G := (λ *val-list*. (f w) (*hd val-list*)))) A **using** *GM* **by** *simp*

from *S Asent* **have** B: *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c g w (c A) =
value-fm κ (λ x. *undefined*) (D, $\mathfrak{C}w$ w, $\mathfrak{F}w$ w, ($\mathfrak{R}w$ w) (G := (λ *val-list*. (g w) (*hd val-list*)))) A **using** *GM* **by** *simp*

show *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c f w (c A) $\leq$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c g w (c A)
using A B *Res* **by** *simp*
qed
qed
from *this* **have** *Res1*: $\forall$ w. $\forall$ $d \in c$‘ *sentences*.
jumpW κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c f w d $\leq$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c g w d **using** *GM* **by** *simp*

have A: $\forall w$ d. $d \notin c$‘ *sentences* $\longrightarrow$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c f w d = *f3* **using** *GM* **by** *simp*
have B: $\forall w$ d. $d \notin c$‘ *sentences* $\longrightarrow$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c g w d = *f3* **using** *GM* **by** *simp*
from A B **have** *Res2*: $\forall$ w d. $d \notin c$‘ *sentences* $\longrightarrow$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c f w d $\leq$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c g w d **by** *simp*

have $\forall$ w d. *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c f w d $\leq$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c g w d
using *Res1 Res2* **by** *smt*
then show *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c f $\leq$ *jumpW* κ (D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G c g
using *le-funI* **by** *smt*
qed

lemma *jumpW-ν-monot*: ‹ *ground-Wmod f4 t4* $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c* $\Longrightarrow f \leq g$
$\Longrightarrow$ *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c f* $\leq$ *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c g*›
proof−
assume *GM*: *ground-Wmod f4 t4* $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c*
then have *S*: *c‘ sentences* $\subseteq D$ **by** *simp*
assume *H*: $f \leq g$
hence $\forall\ w.\ (\mathfrak{R}w\ w)\ (G := (\lambda\ val\text{-}list.\ (f\ w)\ (hd\ val\text{-}list))\ )$
$\leq (\mathfrak{R}w\ w)\ (G := (\lambda\ val\text{-}list.\ (g\ w)\ (hd\ val\text{-}list))\ )$
by (*simp add*: *H le-funD le-funI*)

then have $\forall\ w.$ *leqMod* $(D, \mathfrak{C}w\ w, \mathfrak{F}w\ w, (\mathfrak{R}w\ w)\ (G := (\lambda\ val\text{-}list.\ (f\ w)\ (hd\ val\text{-}list))))$
$(D, \mathfrak{C}w\ w, \mathfrak{F}w\ w, (\mathfrak{R}w\ w)\ (G := (\lambda\ val\text{-}list.\ (g\ w)\ (hd\ val\text{-}list))))$
by *simp*
from *this monot-in-ν* **have** *Res*: $\bigwedge\ s\ A\ w.$
value-fm $\nu\ s\ (D, \mathfrak{C}w\ w, \mathfrak{F}w\ w, (\mathfrak{R}w\ w)\ (G := (\lambda\ val\text{-}list.\ (f\ w)\ (hd\ val\text{-}list))))\ A$
$\leq$ *value-fm* $\nu\ s\ (D, \mathfrak{C}w\ w, \mathfrak{F}w\ w, (\mathfrak{R}w\ w)\ (G := (\lambda\ val\text{-}list.\ (g\ w)\ (hd\ val\text{-}list))))\ A$ **by** *fastforce*

have $\forall\ w.\ \forall\ A \in$ *sentences*.
jumpW ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c f w* $(c\ A) \leq$ *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c g w* $(c\ A)$ **proof**
fix *w*
show $\forall\ A \in$ *sentences*.
jumpW ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c f w* $(c\ A) \leq$ *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c g w* $(c\ A)$ **proof**
fix *A*::('*d*, '*c*, '*e*) *fm*
assume *Asent*: ‹$A \in$ *sentences*›

from *S this* **have** *A*: *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c f w* $(c\ A) =$
value-fm $\nu\ (\lambda\ x.\ undefined)\ (D, \mathfrak{C}w\ w, \mathfrak{F}w\ w, (\mathfrak{R}w\ w)\ (G := (\lambda\ val\text{-}list.\ (f\ w)\ (hd\ val\text{-}list))))\ A$ **using** *GM* **by** *simp*

from *S Asent* **have** *B*: *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c g w* $(c\ A) =$
value-fm $\nu\ (\lambda\ x.\ undefined)\ (D, \mathfrak{C}w\ w, \mathfrak{F}w\ w, (\mathfrak{R}w\ w)\ (G := (\lambda\ val\text{-}list.\ (g\ w)\ (hd\ val\text{-}list))))\ A$ **using** *GM* **by** *simp*

show *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c f w* $(c\ A) \leq$ *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c g w* $(c\ A)$
using *A B Res* **by** *simp*
qed
qed
from *this* **have** *Res1*: $\forall\ w.\ \forall\ d \in$ *c‘ sentences*.
jumpW ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c f w d* $\leq$ *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c g w d* **using** *GM* **by** *simp*

have *A*: $\forall w\ d.\ d \notin$ *c‘ sentences* $\longrightarrow$ *jumpW* ν $(D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G c f w d* $=$ *f4* **using** *GM* **by** *simp*

have *B*: $\forall$ *w d*. *d* $\notin$ *c*' *sentences* $\longrightarrow$ *jumpW* ν (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g w d* = *f4* **using** *GM* **by** *simp*
from *A B* **have** *Res2*: $\forall$ *w d*. *d* $\notin$ *c*' *sentences* $\longrightarrow$ *jumpW* ν (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w d* $\leq$ *jumpW* ν (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g w d* **by** *simp*

have $\forall$ *w d*. *jumpW* ν (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f w d* $\leq$ *jumpW* ν (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g w d*
using *Res1 Res2* **by** *smt*
then show *jumpW* ν (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c f* $\leq$ *jumpW* ν (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g*
using *le-funI* **by** *smt*
qed

lemma μ-*Wfixed-point-prop*:
⟨*ground-Wmod f3 t3* (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c* $\Longrightarrow$
($\exists$ *g*. *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g* = *g*)⟩
proof−
assume *GM*: *ground-Wmod f3 t3* (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c*
then have *H*: *c*' *sentences* $\subseteq$ *D* **by** *simp*
let *?U* = (*UNIV* :: ('*w* $\Rightarrow$ '*v* $\Rightarrow$ *bool3*) *set*)
have *1*: *ccpo ?U*
using *function-space-ccpo-bool3 function-space-ccpo-full* **by** *auto*

have $\forall$ *g1 g2*.
g1 $\leq$ *g2* $\longrightarrow$ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g1*
$\leq$ *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g2*
using *jumpW*-μ-*monot GM* **by** *metis*
then have *2*: *monot* (*jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c*) **by** *simp*
from *1 2 VisserFixp* **have** *ccpo* (*FixPs ?U* (*jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c*))
by *blast*
from *this ccpo-least-element* **have** $\exists$ *g*. *g* $\in$ (*FixPs ?U* (*jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c*)) **by** *auto*
then obtain *g* **where** *Hg*: ⟨*g* $\in$ (*FixPs ?U* (*jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c*))⟩ **by** *auto*
show $\exists$ *g*. *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g* = *g* **proof**
show *jumpW* μ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g* = *g* **using** *FixPs-def Hg* **by** *blast*
qed
qed

lemma κ-*Wfixed-point-prop*:
⟨*ground-Wmod f3 t3* (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c* $\Longrightarrow$
($\exists$ *g*. *jumpW* κ (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c g* = *g*)⟩
proof−
assume *GM*: *ground-Wmod f3 t3* (*D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G c*
then have *H*: *c*' *sentences* $\subseteq$ *D* **by** *simp*
let *?U* = (*UNIV* :: ('*w* $\Rightarrow$ '*v* $\Rightarrow$ *bool3*) *set*)
have *1*: *ccpo ?U*
using *function-space-ccpo-bool3 function-space-ccpo-full* **by** *auto*

have ∀ *g1 g2*.
g1 ≤ *g2* ⟶ *jumpW κ* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c g1*
≤ *jumpW κ* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c g2*
using *jumpW-κ-monot GM* **by** *metis*
then have *2*: *monot* (*jumpW κ* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c*) **by** *simp*
from *1 2 VisserFixp* **have** *ccpo* (*FixPs ?U* (*jumpW κ* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c*))
by *blast*
from *this ccpo-least-element* **have** ∃*g*. *g* ∈ (*FixPs ?U* (*jumpW κ* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c*)) **by** *auto*
then obtain *g* **where** *Hg*: ‹*g* ∈ (*FixPs ?U* (*jumpW κ* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c*))› **by** *auto*
show ∃ *g*. *jumpW κ* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c g* = *g* **proof**
show *jumpW κ* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c g* = *g* **using** *FixPs-def Hg* **by** *blast*
qed
qed

lemma *ν-Wfixed-point-prop*:
‹*ground-Wmod f4 t4* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c* ⟹
(∃ *g*. *jumpW ν* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c g* = *g*)›
proof−
assume *GM*: *ground-Wmod f4 t4* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c*
then have *H*: *c‘ sentences* ⊆ *D* **by** *simp*
let *?U* = (*UNIV* :: (*'w* ⇒ *'v* ⇒ *bool4*) *set*)
have *1*: *ccpo ?U*
using *function-space-ccpo-bool4 function-space-ccpo-full* **by** *auto*

have ∀ *g1 g2*.
g1 ≤ *g2* ⟶ *jumpW ν* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c g1*
≤ *jumpW ν* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c g2*
using *jumpW-ν-monot GM* **by** *metis*
then have *2*: *monot* (*jumpW ν* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c*) **by** *simp*
from *1 2 VisserFixp* **have** *ccpo* (*FixPs ?U* (*jumpW ν* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c*))
by *blast*
from *this ccpo-least-element* **have** ∃*g*. *g* ∈ (*FixPs ?U* (*jumpW ν* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c*)) **by** *auto*
then obtain *g* **where** *Hg*: ‹*g* ∈ (*FixPs ?U* (*jumpW ν* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c*))› **by** *auto*
show ∃ *g*. *jumpW ν* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c g* = *g* **proof**
show *jumpW ν* (*D*, 𝔆*w*, 𝔉*w*, 𝔑*w*) *G c g* = *g* **using** *FixPs-def Hg* **by** *blast*
qed
qed

type-synonym (*'w*, *'v*, *'a*, *'b*, *'c*, *'mybool*) *tpmodel*
= ‹ (*'w* ⇒ *'w set* ⇒ *real*) × *'v set* × (*'w* ⇒ (*'v*, *'a*,*'b*,*'c*) *const-mod*) × (*'w* ⇒ (*'v*, *'a*,*'b*,*'c*) *func-mod*)
× (*'w* ⇒ (*'v*, *'a*,*'b*,*'c*, *'mybool*) *rela-mod*)›

fun *is-probmes*:: ‹(*'w* ⇒ *'w set* ⇒ *real*) ⇒ *bool*› **where**
(*is-probmes m*) = (∀ *w* :: *'w*. (*m w UNIV* = *1*

$\wedge$ ($\forall$ A :: $'w$ set. m w $A \geq 0$) $\wedge$
($\forall$ A B :: $'w$ set. $A \cap B = \{\} \longrightarrow m\ w\ A + m\ w\ B = m\ w\ (A \cup B)$)))

lemma *probmes-subset*: $[\![$ *is-probmes m*; $A \subseteq B$ $]\!] \Longrightarrow m\ w\ A \leq m\ w\ B$
proof−
  assume *H*: *is-probmes m*
  assume *SS*: $A \subseteq B$
  then have *1*: $B = A \cup (B - A)$ **by** *blast*
  have *2*: $A \cap (B - A) = \{\}$ **by** *blast*
  from *H 1 2* **have** *3*: $m\ w\ B = m\ w\ A + m\ w\ (B - A)$ **by** *simp*
  from *H* **have** *4*: $m\ w\ (B - A) \geq 0$ **by** *simp*
  from *3 4* **show** *?thesis* **by** *simp*
qed

fun *ground-tpmod* :: ‹$'mybool \Rightarrow 'mybool \Rightarrow ('w, 'v, 'a, 'b, 'c, 'mybool)$ *tpmodel*
  $\Rightarrow 'c \Rightarrow 'c \Rightarrow (('a, 'b, 'c)\ fm \Rightarrow 'v) \Rightarrow (rat \Rightarrow 'v) \Rightarrow bool$› **where**
ground-tpmod myFalse myTrue $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H c1 c2* =
((*inj c1* $\wedge$ *c1*‘ *sentences* $\subseteq D$) $\wedge$ (*inj c2*) $\wedge$ *is-probmes m* $\wedge$
($\forall$ *w rsymb val-list*. *rsymb* $\notin \{ G, H \} \longrightarrow$
  $\mathfrak{R}w$ *w rsymb val-list* $\in$ {*myTrue*, *myFalse*}))

fun *modpplus* :: ‹$('w, 'v, 'a, 'b, 'c, 'mybool)$ *tpmodel* $\Rightarrow 'c \Rightarrow 'c$
$\Rightarrow ('w \Rightarrow 'v \Rightarrow 'mybool) \Rightarrow ('w \Rightarrow 'v \Rightarrow 'v \Rightarrow 'mybool) \Rightarrow 'w \Rightarrow ('v, 'a, 'b, 'c,$
$'mybool)$ *model*›
  where
modpplus $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w* =
$(D, \mathfrak{C}w\ w, \mathfrak{F}w\ w, (\mathfrak{R}w\ w)$
($G := (\lambda$ *val-list*. $(g\ w)$ (*hd val-list*)), $H := (\lambda$ *val-list*. $(h\ w)$ (*hd val-list*) (*last val-list*))))

fun *pVal-κ* :: $('w, 'v, 'a, 'b, 'c, bool3)$ *tpmodel* $\Rightarrow 'c \Rightarrow 'c$
$\Rightarrow ('w \Rightarrow 'v \Rightarrow bool3) \Rightarrow ('w \Rightarrow 'v \Rightarrow 'v \Rightarrow bool3) \Rightarrow 'w \Rightarrow ('a, 'b, 'c)\ fm \Rightarrow rat$
$\Rightarrow bool3$ **where**
pVal-κ $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w* φ *q* =
(*if* (*m w* {*w1* . *value-fm* κ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ = *t3*} $\geq$ *real-of-rat q*) *then t3 else*
(
if (*m w* {*w1*. *value-fm* κ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ = *f3*} > (*1* − *real-of-rat q*)) *then*
f3 else n3))

lemma *jumptp-κ-monot-helperT*:
 ground-tpmod f3 t3 $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H c1 c2*
  $\Longrightarrow g1 \leq g2 \Longrightarrow h1 \leq h2$
  $\Longrightarrow$ *value-fm* κ (λ *x*. *undefined*) (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*)
$\varphi = t3$
  $\Longrightarrow$ *value-fm* κ (λ *x*. *undefined*) (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*)
$\varphi = t3$
proof−

assume *H1*: (*ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*)
from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: *g1* $\leq$ *g2*
assume *H3*: *h1* $\leq$ *h2*

have $\forall$ *w*. ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g1 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h1 w*) (*hd val-list*) (*last val-list*)))
$\leq$ ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g2 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h2 w*) (*hd val-list*) (*last val-list*)))
by (*simp add*: *H1 H2 H3 le-funD le-funI*)
from *this* **have** $\forall$ *w1*. *leqMod* (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) **by** *simp*
from *this monot-in-*κ **have** *leq-fact*: $\forall$ *w1*. *value-fm* κ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ $\leq$ *value-fm* κ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ **by** *fastforce*
from *leq-fact* **have** *Implfact*: $\forall$ *w1*. *value-fm* κ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *t3* $\longrightarrow$
value-fm κ (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *t3*
by (*smt bool3.exhaust leq3.simps*(*6*) *leq3.simps*(*7*) *less-eq-bool3-def*)
assume *value-fm* κ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *t3*
from *this Implfact* **show** *value-fm* κ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *t3* **by** *simp*
qed

lemma *jumptp-*κ*-monot-helperF*:
ground-tpmod f3 t3 (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*
$\Longrightarrow$ *g1* $\leq$ *g2* $\Longrightarrow$ *h1* $\leq$ *h2*
$\Longrightarrow$ *value-fm* κ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *f3*
$\Longrightarrow$ *value-fm* κ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *f3*
proof−
assume *H1*: (*ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*)
from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: *g1* $\leq$ *g2*
assume *H3*: *h1* $\leq$ *h2*

have $\forall$ *w*. ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g1 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h1 w*) (*hd val-list*) (*last val-list*)))
$\leq$ ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g2 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h2 w*) (*hd val-list*) (*last val-list*)))
by (*simp add*: *H1 H2 H3 le-funD le-funI*)
from *this* **have** $\forall$ *w1*. *leqMod* (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*)

(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g2 h2 w1*) **by** *simp*
  from *this monot-in-κ* **have** *leq-fact*: $\forall$ *w1*. *value-fm* κ (λ x. *undefined*)
(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1 w1*) $\varphi \leq$ *value-fm* κ (λ x. *undefined*)
(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g2 h2 w1*) φ **by** *fastforce*
  from *leq-fact* **have** *Implfact*: $\forall$ *w1*. *value-fm* κ (λ x. *undefined*)
(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1 w1*) φ = *f3* $\longrightarrow$
value-fm κ (λ x. *undefined*)
(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g2 h2 w1*) φ = *f3*
  by (*smt bool3.exhaust leq3.simps less-eq-bool3-def*)
  assume *value-fm* κ (λ x. *undefined*) (*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1 w1*) φ = *f3*
  from *this Implfact* **show** *value-fm* κ (λ x. *undefined*) (*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g2 h2 w1*) φ = *f3* **by** *simp*
qed

lemma *pVal-κ-monot*: $[\![$ (*ground-tpmod f3 t3* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2*);
g1 $\leq$ *g2*; *h1* $\leq$ *h2* $]\!] \Longrightarrow$
pVal-κ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1 w* φ q
$\leq$ *pVal-κ* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g2 h2 w* φ q
proof−
assume *H1*: (*ground-tpmod f3 t3* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2*)
from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: *g1* $\leq$ *g2*
assume *H3*: *h1* $\leq$ *h2*

show *?thesis* **proof**(*cases pVal-κ* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1 w* φ q)
case *n3*
then show *?thesis* **by**(*simp add*: *less-eq-bool3-def*)
next
case *t3*
then have *Ht31*: *pVal-κ* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1 w* φ q = *t3* **by** *auto*
from *Ht31* **have** *Estimate1*: ‹*m w* {*w1* . *value-fm* κ (λ x. *undefined*)
(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1 w1*) φ = *t3*} $\geq$ *real-of-rat q*›
  by (*smt Collect-cong bool3.distinct*(*1*) *bool3.simps*(*6*) *pVal-κ.simps*)
from *jumptp-κ-monot-helperT H1 H2 H3*
 have *Implfact*: $\forall$ *w1*. *value-fm* κ (λ x. *undefined*)
(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1 w1*) φ = *t3* $\longrightarrow$
value-fm κ (λ x. *undefined*)
(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g2 h2 w1*) φ = *t3*
  by (*smt bool3.exhaust leq3.simps*(*6*) *leq3.simps*(*7*) *less-eq-bool3-def*)

  let *?Set1* = {*w1* . *value-fm* κ (λ x. *undefined*)
(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1 w1*) φ = *t3*}
  let *?Set2* = {*w1* . *value-fm* κ (λ x. *undefined*)
(*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g2 h2 w1*) φ = *t3*}
  from *Implfact* **have** *?Set1* $\subseteq$ *?Set2* **by** *auto*
  from *this Hpm probmes-subset*[*of m ?Set1 ?Set2*] **have** *m w ?Set1* $\leq$ *m w ?Set2* **by** *simp*
  from *this Estimate1* **have** *m w ?Set2* $\geq$ *real-of-rat q* **by** *simp*

from *this* **have** *Ht32*: *pVal-κ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w φ q* = *t3* **by** *simp*
from *Ht31 Ht32* **show** *?thesis* **by** *simp*
next
case *f3*
then have *Hf31*: *pVal-κ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w φ q* = *f3* **by** *auto*
from *Hf31* **have** *Estimate2*: ‹*m w* {*w1* . *value-fm κ* (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) *φ* = *f3*} > (*1* − *real-of-rat q*)›
by (*smt Collect-cong bool3.distinct bool3.simps pVal-κ.simps*)
from *jumptp-κ-monot-helperF H1 H2 H3* **have** *Implfact*: ∀ *w1*. *value-fm κ* (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) *φ* = *f3* ⟶
value-fm κ (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) *φ* = *f3*
by (*smt bool3.exhaust leq3.simps less-eq-bool3-def*)

let *?Set1* = {*w1* . *value-fm κ* (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) *φ* = *f3*}
let *?Set2* = {*w1* . *value-fm κ* (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) *φ* = *f3*}
from *Implfact* **have** *?Set1* ⊆ *?Set2* **by** *auto*
from *this Hpm probmes-subset*[*of m ?Set1 ?Set2*] **have** *m w ?Set1* ≤ *m w ?Set2* **by** *simp*
from *this Estimate2* **have** *Cond1*: *m w ?Set2* > *1* − *real-of-rat q* **by** *auto*
from *this* **have** *Cond1B*: *1* < *real-of-rat q* + *m w ?Set2* **by** *simp*

let *?Set2T* = {*w1* . *value-fm κ* (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) *φ* = *t3*}

have *?Set2* ∪ *?Set2T* ⊆ *UNIV* **by** *auto*
from *this Hpm probmes-subset*[*of m ?Set2* ∪ *?Set2T UNIV*]
have *PS1*: *m w* (*?Set2* ∪ *?Set2T*) ≤ *m w UNIV* **by** *simp*
have *?Set2* ∩ *?Set2T* = {} **by** *auto*
from *this Hpm* **have** *PS2*: *m w ?Set2* + *m w ?Set2T* = *m w* (*?Set2* ∪ *?Set2T*) **by** *simp*
from *PS1 PS2* **have** *PS3*: *m w ?Set2* + *m w ?Set2T* ≤ *m w UNIV* **by** *simp*
have *m w UNIV* = *1* **using** *Hpm* **by** *simp*
from *this PS3* **have** *m w ?Set2* + *m w ?Set2T* ≤ *1* **by** *simp*
from *this Cond1* **have** *m w ?Set2* + *m w ?Set2T* < *real-of-rat q* + *m w ?Set2* **by** *simp*
from *this* **have** *Cond2*: *m w ?Set2T* < *real-of-rat q* **by** *simp*
from *Cond1 Cond2* **have** *Hf32*: *pVal-κ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w φ q* = *f3* **by** *simp*
from *Hf31 Hf32* **show** *?thesis* **by** *simp*
qed
qed

fun *pVal-μ* :: (′*w*, ′*v*, ′*a*, ′*b*, ′*c*, *bool3*) *tpmodel* ⇒ ′*c* ⇒ ′*c*
⇒ (′*w* ⇒ ′*v* ⇒ *bool3*) ⇒ (′*w* ⇒ ′*v* ⇒ ′*v* ⇒ *bool3*) ⇒ ′*w* ⇒ (′*a*, ′*b*, ′*c*) *fm* ⇒ *rat*

$\Rightarrow$ *bool3* **where**
pVal-μ $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w* φ *q* =
(*if* (*m w* {*w1* . *value-fm* μ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ = *t3*} $\geq$ *real-of-rat q*
$\wedge$ *m w* {*w1* . *value-fm* μ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ = *n3*} = *0*) *then t3 else* (
if (*m w* {*w1*. *value-fm* μ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ = *f3*} > (*1* − *real-of-rat q*)
$\wedge$ *m w* {*w1* . *value-fm* μ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ = *n3*} = *0*) *then f3 else n3*))

lemma *jumptp-μ-monot-helperT*:
ground-tpmod f3 t3 $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H c1 c2*
$\Longrightarrow$ *g1* $\leq$ *g2* $\Longrightarrow$ *h1* $\leq$ *h2*
$\Longrightarrow$ *value-fm* μ (λ *x*. *undefined*) (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*)
φ = *t3*
$\Longrightarrow$ *value-fm* μ (λ *x*. *undefined*) (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*)
φ = *t3*
proof−
assume *H1*: (*ground-tpmod f3 t3* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H c1 c2*)
from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: *g1* $\leq$ *g2*
assume *H3*: *h1* $\leq$ *h2*

have $\forall$ *w*. ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g1 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h1 w*) (*hd val-list*) (*last val-list*)))
$\leq$ ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g2 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h2 w*) (*hd val-list*) (*last val-list*)))
by (*simp add*: *H1 H2 H3 le-funD le-funI*)
from *this* **have** $\forall$ *w1*. *leqMod* (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*) **by** *simp*
from *this monot-in-μ* **have** *leq-fact*: $\forall$ *w1*. *value-fm* μ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*) φ $\leq$ *value-fm* μ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*) φ **by** *fastforce*
from *leq-fact* **have** *Implfact*: $\forall$ *w1*. *value-fm* μ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*) φ = *t3* $\longrightarrow$
value-fm μ (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*) φ = *t3*
by (*smt bool3.exhaust leq3.simps*(*6*) *leq3.simps*(*7*) *less-eq-bool3-def*)
assume *value-fm* μ (λ *x*. *undefined*) (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*) φ = *t3*
from *this Implfact* **show** *value-fm* μ (λ *x*. *undefined*) (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*) φ = *t3* **by** *simp*
qed

lemma *jumptp-μ-monot-helperF*:
ground-tpmod f3 t3 $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H c1 c2*

$\Longrightarrow$ *g1* $\leq$ *g2* $\Longrightarrow$ *h1* $\leq$ *h2*
$\Longrightarrow$ *value-fm* μ (λ *x. undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *f3*
$\Longrightarrow$ *value-fm* μ (λ *x. undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *f3*
proof−
assume *H1*: (*ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*)
from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: *g1* $\leq$ *g2*
assume *H3*: *h1* $\leq$ *h2*

have $\forall$ *w*. ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g1 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h1 w*) (*hd val-list*) (*last val-list*)))
$\leq$ ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g2 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h2 w*) (*hd val-list*) (*last val-list*)))
by (*simp add*: *H1 H2 H3 le-funD le-funI*)
from *this* **have** $\forall$ *w1*. *leqMod* (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) **by** *simp*
from *this monot-in-*μ **have** *leq-fact*: $\forall$ *w1*. *value-fm* μ (λ *x. undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ $\leq$ *value-fm* μ (λ *x. undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ **by** *fastforce*
from *leq-fact* **have** *Implfact*: $\forall$ *w1*. *value-fm* μ (λ *x. undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *f3* $\longrightarrow$
value-fm μ (λ *x. undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *f3*
by (*smt bool3.exhaust leq3.simps less-eq-bool3-def*)
assume *value-fm* μ (λ *x. undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *f3*
from *this Implfact* **show** *value-fm* μ (λ *x. undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *f3* **by** *simp*
qed

lemma *pVal-*μ*-monot*: $[\![$ (*ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*);
g1 $\leq$ *g2*; *h1* $\leq$ *h2* $]\!] \Longrightarrow$
pVal-μ (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w* φ *q*
$\leq$ *pVal-*μ (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w* φ *q*
proof−
assume *H1*: (*ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*)
from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: *g1* $\leq$ *g2*
assume *H3*: *h1* $\leq$ *h2*

show *?thesis* **proof**(*cases pVal-*μ (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w* φ *q*)
case *n3*
then show *?thesis* **by**(*simp add*:*less-eq-bool3-def*)
next
case *t3*

from *t3* **have** *Estimate1*: $m\ w\ \{w1\ .\ value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi = t3\} \geq real\text{-}of\text{-}rat\ q$
by (*smt Collect-cong bool3.distinct bool3.simps pVal-μ.simps*)

let $?NSet1 = \{w1\ .\ value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi = n3\}$
let $?NSet2 = \{w1\ .\ value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi = n3\}$

from *t3* **have** *Estimate2*: $m\ w\ ?NSet1 = 0$
by (*smt Collect-cong bool3.distinct bool3.simps pVal-μ.simps*)
have $\forall\ w1.$
$value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi = n3$
$\longrightarrow value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi = n3$ **proof**
fix *w1*
show $value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi = n3$
$\longrightarrow value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi = n3$
proof(*cases* $value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi$)
case *n3*
then show *?thesis* **by** *simp*
next
case *t3*
from *this jumptp-μ-monot-helperT H1 H2 H3* **have** ‹$value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi = t3$› **by** *metis*
then show *?thesis* **by** *simp*
next
case *f3*
from *this jumptp-μ-monot-helperF H1 H2 H3* **have** ‹$value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi = f3$› **by** *metis*
then show *?thesis* **by** *simp*
qed
qed
hence $?NSet2 \subseteq ?NSet1$ **by** *auto*
from *this Hpm probmes-subset*[*of m ?NSet2 ?NSet1*]
have ‹$m\ w\ ?NSet2 \leq m\ w\ ?NSet1$› **by** *simp*
from *this Estimate2* **have** *A*: $m\ w\ ?NSet2 \leq 0$ **by** *simp*
from *Hpm* **have** *B*: $m\ w\ ?NSet2 \geq 0$ **by** *simp*
from *A B* **have** *Estimate3*: $m\ w\ ?NSet2 = 0$ **by** *simp*

from *jumptp-μ-monot-helperT H1 H2 H3*
have *Implfact*: $\forall\ w1.\ value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi = t3 \longrightarrow$
$value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi = t3$

by (*smt bool3.exhaust leq3.simps*(*6*) *leq3.simps*(*7*) *less-eq-bool3-def*)

let *?Set1* = {*w1* . *value-fm* μ (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *t3*}
let *?Set2* = {*w1* . *value-fm* μ (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *t3*}
from *Implfact* **have** *?Set1* ⊆ *?Set2* **by** *auto*
from *this Hpm probmes-subset*[*of m ?Set1 ?Set2*] **have** *m w ?Set1* ≤ *m w ?Set2*
by *simp*
from *this Estimate1* **have** *Estimate4*: *m w ?Set2* ≥ *real-of-rat q* **by** *simp*
from *Estimate3 Estimate4* **have** ‹*pVal-*μ (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w* φ
q = *t3*› **by** *simp*
then show *?thesis* **using** *t3* **by** *simp*
next
case *f3*
then have *Estimate2*: ‹*m w* {*w1* . *value-fm* μ (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *f3*} > (*1* − *real-of-rat q*)›
by (*smt Collect-cong bool3.distinct bool3.simps pVal-*μ*.simps*)
from *jumptp-*μ*-monot-helperF H1 H2 H3* **have** *Implfact*: ∀ *w1*. *value-fm* μ (λ
x. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *f3* ⟶
value-fm μ (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *f3*
by (*smt bool3.exhaust leq3.simps less-eq-bool3-def*)

let *?Set1* = {*w1* . *value-fm* μ (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *f3*}
let *?Set2* = {*w1* . *value-fm* μ (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *f3*}
from *Implfact* **have** *?Set1* ⊆ *?Set2* **by** *auto*
from *this Hpm probmes-subset*[*of m ?Set1 ?Set2*] **have** *m w ?Set1* ≤ *m w ?Set2*
by *simp*
from *this Estimate2* **have** *Cond1*: *m w ?Set2* > *1* − *real-of-rat q* **by** *auto*
from *this* **have** *Cond1B*: *1* < *real-of-rat q* + *m w ?Set2* **by** *simp*

let *?Set2T* = {*w1* . *value-fm* μ (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *t3*}

have *?Set2* ∪ *?Set2T* ⊆ *UNIV* **by** *auto*
from *this Hpm probmes-subset*[*of m ?Set2* ∪ *?Set2T UNIV*]
have *PS1*: *m w* (*?Set2* ∪ *?Set2T*) ≤ *m w UNIV* **by** *simp*
have *?Set2* ∩ *?Set2T* = {} **by** *auto*
from *this Hpm* **have** *PS2*: *m w ?Set2* + *m w ?Set2T* = *m w* (*?Set2* ∪ *?Set2T*)
by *simp*
from *PS1 PS2* **have** *PS3*: *m w ?Set2* + *m w ?Set2T* ≤ *m w UNIV* **by** *simp*
have *m w UNIV* = *1* **using** *Hpm* **by** *simp*
from *this PS3* **have** *m w ?Set2* + *m w ?Set2T* ≤ *1* **by** *simp*
from *this Cond1* **have** *m w ?Set2* + *m w ?Set2T* < *real-of-rat q* + *m w ?Set2*
by *simp*

from *this* **have** *Cond2*: $m\ w\ ?Set2T < real\text{-}of\text{-}rat\ q$ **by** *simp*

let *?NSet1* = $\{w1\ .\ value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi = n3\}$
let *?NSet2* = $\{w1\ .\ value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi = n3\}$

from *f3* **have** *Estimate2*: $m\ w\ ?NSet1 = 0$
by (*smt Collect-cong bool3.distinct bool3.simps pVal-μ.simps*)
have $\forall\ w1.$
$value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi =$
$n3$
$\longrightarrow value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)$
$\varphi = n3$ **proof**
fix *w1*
show $value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2$
$w1)\ \varphi = n3$
$\longrightarrow value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)$
$\varphi = n3$
proof(*cases* $value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H$
$g1\ h1\ w1)\ \varphi)$
case *n3*
then show *?thesis* **by** *simp*
next
case *t3*
from *this jumptp-μ-monot-helperT H1 H2 H3* **have** ‹$value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi = t3$› **by** *metis*
then show *?thesis* **by** *simp*
next
case *f3*
from *this jumptp-μ-monot-helperF H1 H2 H3* **have** ‹$value\text{-}fm\ \mu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi = f3$› **by** *metis*
then show *?thesis* **by** *simp*
qed
qed
hence $?NSet2 \subseteq ?NSet1$ **by** *auto*
from *this Hpm probmes-subset*[*of m ?NSet2 ?NSet1*]
have ‹$m\ w\ ?NSet2 \leq m\ w\ ?NSet1$› **by** *simp*
from *this Estimate2* **have** *A*: $m\ w\ ?NSet2 \leq 0$ **by** *simp*
from *Hpm* **have** *B*: $m\ w\ ?NSet2 \geq 0$ **by** *simp*
from *A B* **have** *Estimate3*: $m\ w\ ?NSet2 = 0$ **by** *simp*

from *Estimate3 Cond1 Cond2* **have** ‹$pVal\text{-}\mu\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w$
$\varphi\ q = f3$› **by** *simp*
then show *?thesis* **using** *f3* **by** *simp*
qed
qed

fun *pVal-ν* :: $('w,\ 'v,\ 'a,\ 'b,\ 'c,\ bool4)\ tpmodel \Rightarrow 'c \Rightarrow 'c$

$\Rightarrow ('w \Rightarrow 'v \Rightarrow bool4) \Rightarrow ('w \Rightarrow 'v \Rightarrow 'v \Rightarrow bool4) \Rightarrow 'w \Rightarrow ('a, 'b, 'c)\ fm \Rightarrow rat$
$\Rightarrow bool4$ **where**
pVal-ν $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w φ q* =
(*if* (*m w* {*w1* . *value-fm ν* (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ ≥ *t4*} ≥ *real-of-rat q*
∧ *m w* {*w1* . *value-fm ν* (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ ≥ *f4*} > (*1* − *real-of-rat q*)) *then*
b4
else (
if (*m w* {*w1*. *value-fm ν* (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ ≥ *t4*} ≥ *real-of-rat q*) *then t4*
else (*if* (*m w* {*w1* . *value-fm ν* (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g h w1*) φ ≥ *f4*} > (*1* − *real-of-rat q*)) *then*
f4
else n4)))

lemma *jumptp-ν-monot-helperB*:
ground-tpmod f4 t4 $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H c1 c2*
⟹ *g1* ≤ *g2* ⟹ *h1* ≤ *h2*
⟹ *value-fm ν* (λ *x*. *undefined*) (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*)
φ = *b4*
⟹ *value-fm ν* (λ *x*. *undefined*) (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*)
φ = *b4*
proof−
assume *H1*: (*ground-tpmod f4 t4* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H c1 c2*)
from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: *g1* ≤ *g2*
assume *H3*: *h1* ≤ *h2*

have ∀ *w*. ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g1 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h1 w*) (*hd val-list*) (*last val-list*)))
≤ ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g2 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h2 w*) (*hd val-list*) (*last val-list*)))
by (*simp add*: *H1 H2 H3 le-funD le-funI*)
from *this* **have** ∀ *w1*. *leqMod* (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*) **by** *simp*
from *this monot-in-ν* **have** *leq-fact*: ∀ *w1*. *value-fm ν* (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*) φ ≤ *value-fm ν* (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*) φ **by** *fastforce*
from *leq-fact* **have** *Implfact*: ∀ *w1*. *value-fm ν* (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*) φ = *b4* ⟶
value-fm ν (λ *x*. *undefined*)
(*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g2 h2 w1*) φ = *b4*
by (*smt bool4.exhaust leq4.simps less-eq-bool4-def*)
assume *value-fm ν* (λ *x*. *undefined*) (*modpplus* $(m, D, \mathfrak{C}w, \mathfrak{F}w, \mathfrak{R}w)$ *G H g1 h1 w1*) φ = *b4*
from *this Implfact* **show** *value-fm ν* (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$,

$\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ = *b4* **by** *simp*
qed

lemma *jumptp-ν-monot-helperT*:
ground-tpmod f4 t4 (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*
$\Longrightarrow$ *g1* $\leq$ *g2* $\Longrightarrow$ *h1* $\leq$ *h2*
$\Longrightarrow$ *value-fm* ν (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *t4*
$\Longrightarrow$ *value-fm* ν (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) $\varphi \geq$ *t4*
proof−
assume *H1*: (*ground-tpmod f4 t4* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*)
from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: *g1* $\leq$ *g2*
assume *H3*: *h1* $\leq$ *h2*

have ∀ *w*. ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g1 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h1 w*) (*hd val-list*) (*last val-list*)))
$\leq$ ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g2 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h2 w*) (*hd val-list*) (*last val-list*)))
by (*simp add*: *H1 H2 H3 le-funD le-funI*)
from *this* **have** ∀ *w1*. *leqMod* (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) **by** *simp*
from *this monot-in-ν* **have** *leq-fact*: ∀ *w1*. *value-fm* ν (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) $\varphi \leq$ *value-fm* ν (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) φ **by** *fastforce*
from *leq-fact* **have** *Implfact*: ∀ *w1*. *value-fm* ν (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *t4* $\longrightarrow$
value-fm ν (λ *x*. *undefined*)
(*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) $\varphi \geq$ *t4*
by (*smt*)
assume *value-fm* ν (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *t4*
from *this Implfact* **show** *value-fm* ν (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) $\varphi \geq$ *t4* **by** *simp*

qed

lemma *jumptp-ν-monot-helperF*:
ground-tpmod f4 t4 (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*
$\Longrightarrow$ *g1* $\leq$ *g2* $\Longrightarrow$ *h1* $\leq$ *h2*
$\Longrightarrow$ *value-fm* ν (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w1*) φ = *f4*
$\Longrightarrow$ *value-fm* ν (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w1*) $\varphi \geq$ *f4*
proof−
assume *H1*: (*ground-tpmod f4 t4* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*)

from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: $g1 \leq g2$
assume *H3*: $h1 \leq h2$

have $\forall\ w.\ (\mathfrak{R}w\ w)$
$(G := (\lambda\ val\text{-}list.\ (g1\ w)\ (hd\ val\text{-}list)),\ H := (\lambda\ val\text{-}list.\ (h1\ w)\ (hd\ val\text{-}list)\ (last\ val\text{-}list))\ )$
$\leq (\mathfrak{R}w\ w)$
$(G := (\lambda\ val\text{-}list.\ (g2\ w)\ (hd\ val\text{-}list)),\ H := (\lambda\ val\text{-}list.\ (h2\ w)\ (hd\ val\text{-}list)\ (last\ val\text{-}list))\ )$
by (*simp add*: *H1 H2 H3 le-funD le-funI*)
from *this* **have** $\forall\ w1.\ leqMod\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)$ **by** *simp*
from *this monot-in-ν* **have** *leq-fact*: $\forall\ w1.\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi \leq value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi$ **by** *fastforce*
from *leq-fact* **have** *Implfact*: $\forall\ w1.\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi = f4 \longrightarrow$
$value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi \geq f4$
by (*smt*)
assume $value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi = f4$
from *this Implfact* **show** $value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi \geq f4$ **by** *simp*

qed

lemma *pVal-ν-monot*: $[\![\ (ground\text{-}tpmod\ f4\ t4\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ c1\ c2);$
$g1 \leq g2;\ h1 \leq h2\ ]\!] \Longrightarrow$
$pVal\text{-}\nu\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w\ \varphi\ q$
$\leq pVal\text{-}\nu\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w\ \varphi\ q$
proof−
assume *H1*: $(ground\text{-}tpmod\ f4\ t4\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ c1\ c2)$
from *this* **have** *Hpm*: *is-probmes m* **by** *auto*
assume *H2*: $g1 \leq g2$
assume *H3*: $h1 \leq h2$

show *?thesis* **proof**(*cases* $pVal\text{-}\nu\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w\ \varphi\ q$)
case *n4*
then show *?thesis* **by**(*simp add*:*less-eq-bool4-def*)
next
case *t4*
from *t4* **have** *Estimate1*: $m\ w\ \{w1\ .\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)\ (modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi \geq t4\} \geq real\text{-}of\text{-}rat\ q$
by (*smt Collect-cong bool4.distinct bool4.simps pVal-ν.simps*)

from *jumptp-ν-monot-helperT jumptp-ν-monot-helperB H1 H2 H3*
have *Implfact*: $\forall\ w1.\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$

($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g1$ $h1$ $w1$) $\varphi \geq t4 \longrightarrow$
value-fm ν (λ x. *undefined*)
($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ $w1$) $\varphi \geq t4$
by (*smt bool4.exhaust leq4.simps less-eq-bool4-def*)

let *?Set1* = {*w1* . *value-fm* ν (λ x. *undefined*)
($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g1$ $h1$ $w1$) $\varphi \geq t4$}
let *?Set2* = {*w1* . *value-fm* ν (λ x. *undefined*)
($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ $w1$) $\varphi \geq t4$}
from *Implfact* **have** *?Set1* $\subseteq$ *?Set2* **by** *auto*
from *this Hpm probmes-subset*[*of m ?Set1 ?Set2*] **have** *m w ?Set1* $\leq$ *m w ?Set2* **by** *simp*
from *this Estimate1* **have** *Estimate4*: *m w ?Set2* $\geq$ *real-of-rat q* **by** *simp*
from *Estimate4* **have** ‹*pVal-*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ w φ $q \in \{t4, b4\}$› **by** *simp*
from *this* **have** ‹*pVal-*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ w φ $q \geq t4$›
by (*smt bool4.distinct*(*1*) *bool4.simps*(*10*) *bool4.simps*(*12*) *bool4.simps*(*6*) *bool4.simps*(*8*) *insert-iff leq4.elims*(*3*) *less-eq-bool4-def singletonD*)
then show *?thesis* **using** *t4* **by** *simp*
next
case *f4*
then have *Estimate2*: ‹*m w* {*w1* . *value-fm* ν (λ x. *undefined*)
($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g1$ $h1$ $w1$) $\varphi \geq f4$} > (*1* − *real-of-rat q*)›
by (*smt Collect-cong bool4.distinct bool4.simps pVal-*ν*.simps*)
from *jumptp-*ν*-monot-helperF jumptp-*ν*-monot-helperB*
H1 H2 H3 **have** *Implfact*: $\forall$ *w1*. *value-fm* ν (λ x. *undefined*)
($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g1$ $h1$ $w1$) $\varphi \geq f4 \longrightarrow$
value-fm ν (λ x. *undefined*)
($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ $w1$) $\varphi \geq f4$
by (*smt bool4.exhaust leq4.simps less-eq-bool4-def*)

let *?Set1* = {*w1* . *value-fm* ν (λ x. *undefined*)
($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g1$ $h1$ $w1$) $\varphi \geq f4$}
let *?Set2* = {*w1* . *value-fm* ν (λ x. *undefined*)
($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ $w1$) $\varphi \geq f4$}
from *Implfact* **have** *?Set1* $\subseteq$ *?Set2* **by** *auto*
from *this Hpm probmes-subset*[*of m ?Set1 ?Set2*] **have** *m w ?Set1* $\leq$ *m w ?Set2* **by** *simp*
from *this Estimate2* **have** *Cond1*: *m w ?Set2* > *1* − *real-of-rat q* **by** *auto*
from *this* **have** *Cond1B*: *1* < *real-of-rat q* + *m w ?Set2* **by** *simp*

then have ‹*m w* {*w1* . *value-fm* ν (λ x. *undefined*)
($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ $w1$) $\varphi \geq f4$} > (*1* − *real-of-rat q*)› **by** *simp*
from *this* **have** ‹*pVal-*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ w φ $q \in \{f4, b4\}$› **by** *simp*
from *this* **have** ‹*pVal-*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ w φ $q \geq f4$›
by (*smt bool4.distinct*(*1*) *bool4.simps insert-iff leq4.elims*(*3*) *less-eq-bool4-def singletonD*)

from *this f4* **show** *?thesis* **by** *simp*
next
case *b4*
then have *Estimate1*: $m\ w\ \{w1\ .\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi \geq t4\} \geq real\text{-}of\text{-}rat\ q$
by (*smt Collect-cong bool4.distinct bool4.simps pVal-ν.simps*)

from *jumptp-ν-monot-helperT jumptp-ν-monot-helperB H1 H2 H3*
have *Implfact*: $\forall\ w1.\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi \geq t4 \longrightarrow$
$value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi \geq t4$
by (*smt bool4.exhaust leq4.simps less-eq-bool4-def*)

let $?Set1 = \{w1\ .\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi \geq t4\}$
let $?Set2 = \{w1\ .\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi \geq t4\}$
from *Implfact* **have** $?Set1 \subseteq ?Set2$ **by** *auto*
from *this Hpm probmes-subset*[*of m ?Set1 ?Set2*] **have** $m\ w\ ?Set1 \leq m\ w\ ?Set2$ **by** *simp*
from *this Estimate1* **have** *Estimate4*: $m\ w\ ?Set2 \geq real\text{-}of\text{-}rat\ q$ **by** *simp*
from *Estimate4* **have** ‹$pVal\text{-}\nu\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w\ \varphi\ q \in \{t4,\ b4\}$› **by** *simp*
from *this* **have** *GreqP1*: ‹$pVal\text{-}\nu\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w\ \varphi\ q \geq t4$›
by (*smt bool4.distinct(1) bool4.simps(10) bool4.simps(12) bool4.simps(6) bool4.simps(8) insert-iff leq4.elims(3) less-eq-bool4-def singletonD*)

from *b4* **have** *Estimate2*: ‹$m\ w\ \{w1\ .\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi \geq f4\} > (1 - real\text{-}of\text{-}rat\ q)$›
by (*smt Collect-cong bool4.distinct bool4.simps pVal-ν.simps*)
from *jumptp-ν-monot-helperF jumptp-ν-monot-helperB*
H1 H2 H3 **have** *Implfact*: $\forall\ w1.\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi \geq f4 \longrightarrow$
$value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi \geq f4$
by (*smt bool4.exhaust leq4.simps less-eq-bool4-def*)

let $?Set1 = \{w1\ .\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g1\ h1\ w1)\ \varphi \geq f4\}$
let $?Set2 = \{w1\ .\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$
$(modpplus\ (m,\ D,\ \mathfrak{C}w,\ \mathfrak{F}w,\ \mathfrak{R}w)\ G\ H\ g2\ h2\ w1)\ \varphi \geq f4\}$
from *Implfact* **have** $?Set1 \subseteq ?Set2$ **by** *auto*
from *this Hpm probmes-subset*[*of m ?Set1 ?Set2*] **have** $m\ w\ ?Set1 \leq m\ w\ ?Set2$ **by** *simp*
from *this Estimate2* **have** *Cond1*: $m\ w\ ?Set2 > 1 - real\text{-}of\text{-}rat\ q$ **by** *auto*
from *this* **have** *Cond1B*: $1 < real\text{-}of\text{-}rat\ q + m\ w\ ?Set2$ **by** *simp*

then have ‹$m\ w\ \{w1\ .\ value\text{-}fm\ \nu\ (\lambda\ x.\ undefined)$

($modpplus$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ $w1$) $\varphi \geq f4$} $>$ ($1 - real\text{-}of\text{-}rat$ q)›
by *simp*
from *this* **have** ‹$pVal\text{-}\nu$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ w φ $q \in \{f4, b4\}$› **by** *simp*
from *this* **have** *GreqP2*: ‹$pVal\text{-}\nu$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ w φ $q \geq f4$›
by (*smt bool4.distinct*(*1*) *bool4.simps insert-iff leq4.elims*(*3*) *less-eq-bool4-def singletonD*)
from *GreqP2 GreqP1* **have** ‹$pVal\text{-}\nu$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ w φ $q = b4$›
by (*smt bool4.exhaust leq4.simps*(*5*) *leq4.simps*(*7*) *leq4.simps*(*9*) *less-eq-bool4-def*)
then show *?thesis* **using** *b4* **by** *simp*
qed
qed

function *jumptp*κ
where
jumptpκ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $c1$ $c2$ (g,h) =
(*if* (*ground-tpmod f3 t3* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $c1$ $c2$) *then*
((λ w A. *if* ($A \in c1$‘ *sentences*) *then*
value-fm κ (λ x. *undefined*) (*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H g h w) (*inv c1* A)
else f3),
(λ w $v1$ $v2$. *if* ($v1 \in c1$‘*sentences* $\wedge$ $v2 \in$ *range c2*) *then*
$pVal\text{-}\kappa$ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H g h w (*inv c1 v1*) (*inv c2 v2*)
else f3))
else undefined)
apply *auto*[*1*]
by *blast*
termination by *lexicographic-order*

lemma *jumptp-*κ*-monot*: ‹ *ground-tpmod f3 t3* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $c1$ $c2$
$\Longrightarrow g1 \leq g2 \Longrightarrow h1 \leq h2$
$\Longrightarrow$ *fst* (*jumptp*κ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $c1$ $c2$ ($g1$,$h1$)) $\leq$
fst (*jumptp*κ (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $c1$ $c2$ ($g2$,$h2$)) ›
proof−
assume *GM*: *ground-tpmod f3 t3* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $c1$ $c2$
then have *S*: $c1$‘ *sentences* $\subseteq D$ **by** *simp*
assume *H1*: $g1 \leq g2$
assume *H2*: $h1 \leq h2$
have $\forall$ w. ($\mathfrak{R}w$ w)
(G := (λ *val-list*. ($g1$ w) (*hd val-list*)), H := (λ *val-list*. ($h1$ w) (*hd val-list*) (*last val-list*)))
$\leq$ ($\mathfrak{R}w$ w)
(G := (λ *val-list*. ($g2$ w) (*hd val-list*)), H := (λ *val-list*. ($h2$ w) (*hd val-list*) (*last val-list*)))
by (*simp add*: *H1 H2 le-funD le-funI*)

then have $\forall$ w. *leqMod* (*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g1$ $h1$ w) (*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H $g2$ $h2$ w)

by *simp*
from *this monot-in-κ* **have** *Res*: ⋀ *s A w*.
value-fm κ s (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w*) *A*
≤ *value-fm κ s* (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w*) *A* **by** *fastforce*

have ∀ *w*. ∀ *A* ∈ *sentences*.
fst (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w* (*c1 A*) ≤
fst (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w* (*c1 A*) **proof**
fix *w*
show ∀ *A* ∈ *sentences*.
fst (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w* (*c1 A*) ≤
fst (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w* (*c1 A*) **proof**
fix *A*::('*d*, '*c*, '*e*) *fm*
assume *Asent*: ‹*A* ∈ *sentences*›

from *S this* **have** *A*: *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w* (*c1 A*) =
value-fm κ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w*) *A*
using *GM* **by** *simp*

from *S Asent* **have** *B*: *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w* (*c1 A*) =
value-fm κ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w*) *A*
using *GM* **by** *simp*

show *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w* (*c1 A*) ≤
fst (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w* (*c1 A*)
using *A B Res* **by** *simp*
qed
qed
from *this* **have** *Res1*: ∀ *w*. ∀ *d* ∈ *c1*' *sentences*.
fst (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w d* ≤ *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w d* **using** *GM* **by** *simp*

have *A*: ∀ *w d*. *d* ∉ *c1*' *sentences* ⟶ *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w d* = *f3* **using** *GM* **by** *simp*
have *B*: ∀ *w d*. *d* ∉ *c1*' *sentences* ⟶ *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w d* = *f3* **using** *GM* **by** *simp*
from *A B* **have** *Res2*: ∀ *w d*. *d* ∉ *c1*' *sentences* ⟶ *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w d* ≤ *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w d* **by** *simp*

have ∀ *w d*. *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w d* ≤ *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w d*
using *Res1 Res2* **by** *smt*
then show *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) ≤ *fst* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*))
using *le-funI* **by** *smt*
qed

lemma *jumptp-κ-monot2*: ‹ *ground-tpmod f3 t3* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*
⟹ *g1* ≤ *g2* ⟹ *h1* ≤ *h2*
⟹ *snd* (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g1*,*h1*)) ≤
snd (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g2*,*h2*)) ›
proof−
assume *GM*: *ground-tpmod f3 t3* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*
assume *H1*: *g1* ≤ *g2*
assume *H2*: *h1* ≤ *h2*

from *GM H1 H2* **have** *1*: ∀ *w v1 v2*. *v1* ∈ *c1*‘*sentences* ∧ *v2* ∈ *range c2* ⟶
snd (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g1*,*h1*)) *w v1 v2* ≤
snd (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g2*,*h2*)) *w v1 v2*
using *pVal-κ-monot* **by** (*smt jumptpκ.simps snd-conv*)

from *GM* **have** *2*: ∀ *w v1 v2*. *v1* ∉ *c1*‘*sentences* ∨ *v2* ∉ *range c2* ⟶
snd (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g1*,*h1*)) *w v1 v2* ≤
snd (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g2*,*h2*)) *w v1 v2* **by** *simp*
from *1 2* **show** *?thesis* **using** *le-funI* **by** *smt*
qed

lemma *κtp-fixed-point-prop*:
‹*ground-tpmod f3 t3* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* ⟹
(∃ *g h*. *jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g*,*h*) = (*g*,*h*))›
proof−
assume *GM*: *ground-tpmod f3 t3* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*
then have *H*: *c1*‘ *sentences* ⊆ *D* **by** *simp*
let *?U1* = (*UNIV* :: (′*w* ⇒ ′*v* ⇒ *bool3*) *set*)
have *1*: *ccpo ?U1*
using *function-space-ccpo-bool3 function-space-ccpo-full* **by** *auto*

let *?U2* = (*UNIV* :: (′*w* ⇒ ′*v* ⇒ ′*v* ⇒ *bool3*) *set*)
have *2*: *ccpo ?U2*
using *function-space-ccpo-bool3 function-space-ccpo-full* **by** *metis*

let *?U* = *?U1* × *?U2*
from *1 2* **have** *3*: *ccpo ?U* **using** *product-ccpo* **by** *metis*

have ∀ *g1 g2 h1 h2*.
g1 ≤ *g2* ⟶ *h1* ≤ *h2* ⟶ *jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g1*,*h1*)
≤ *jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g2*,*h2*)
using *jumptp-κ-monot jumptp-κ-monot2 GM* **by** (*metis less-eq-prod-def*)
then have *4*: *monot* (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*) **by** *simp*
have *5*: (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*) ‘ *?U* ⊆ *?U* **by** *simp*
from *3 4 5 VisserFixp* **have** *ccpo* (*FixPs ?U* (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*))
by *blast*
from *this ccpo-least-element* **have** ∃ *g h*. (*g*,*h*) ∈ (*FixPs ?U* (*jumptpκ* (*m*, *D*, ℭ*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*)) **by** *fast*

then obtain *g h* **where** *Hg*: ‹$(g,h) \in$ (*FixPs ?U* (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*))› **by** *auto*
show ∃ *g h*. (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*) $(g,h) = (g,h)$ **proof**
show ∃ *h*. (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*) $(g,h) = (g,h)$ **proof**
show (*jumptpκ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*) $(g, h) = (g, h)$ **using** *FixPs-def Hg* **by** *blast*
qed
qed
qed

function *jumptpμ*
where
jumptpμ (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (g,h) =
(*if* (*ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*) *then*
((λ *w A*. *if* (*A* ∈ *c1*‘ *sentences*) *then*
value-fm μ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g h w*) (*inv c1 A*)
else f3),
(λ *w v1 v2*. *if* (*v1* ∈ *c1*‘*sentences* ∧ *v2* ∈ *range c2*) *then*
pVal-μ (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g h w* (*inv c1 v1*) (*inv c2 v2*)
else f3))
else undefined)
apply *auto*[*1*]
by *blast*
termination by *lexicographic-order*

lemma *jumptp-μ-monot*: ‹ *ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*
⟹ *g1* ≤ *g2* ⟹ *h1* ≤ *h2*
⟹ *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) ≤
fst (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) ›
proof−
assume *GM*: *ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*
then have *S*: *c1*‘ *sentences* ⊆ *D* **by** *simp*
assume *H1*: *g1* ≤ *g2*
assume *H2*: *h1* ≤ *h2*
have ∀ *w*. ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g1 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h1 w*) (*hd val-list*) (*last val-list*)))
≤ ($\mathfrak{R}w$ *w*)
(*G* := (λ *val-list*. (*g2 w*) (*hd val-list*)), *H* := (λ *val-list*. (*h2 w*) (*hd val-list*) (*last val-list*)))
by (*simp add*: *H1 H2 le-funD le-funI*)

then have ∀ *w*. *leqMod* (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w*)
by *simp*
from *this monot-in-μ* **have** *Res*: ⋀ *s A w*.
value-fm μ *s* (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w*) *A*
≤ *value-fm* μ *s* (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w*) *A* **by** *fastforce*

have ∀ *w*. ∀ *A* ∈ *sentences*.
fst (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w* (*c1 A*) ≤
fst (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w* (*c1 A*) **proof**
fix *w*
show ∀ *A* ∈ *sentences*.
fst (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w* (*c1 A*) ≤
fst (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w* (*c1 A*) **proof**
fix *A*::(′*d*, ′*c*, ′*e*) *fm*
assume *Asent*: ‹*A* ∈ *sentences*›

from *S this* **have** *A*: *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w* (*c1 A*) =
value-fm μ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g1 h1 w*) *A*
using *GM* **by** *simp*

from *S Asent* **have** *B*: *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w* (*c1 A*) =
value-fm μ (λ *x*. *undefined*) (*modpplus* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H g2 h2 w*) *A*
using *GM* **by** *simp*

show *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w* (*c1 A*) ≤
fst (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w* (*c1 A*)
using *A B Res* **by** *simp*
qed
qed
from *this* **have** *Res1*: ∀ *w*. ∀ *d* ∈ *c1*‘ *sentences*.
fst (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w d* ≤ *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w d* **using** *GM* **by** *simp*

have *A*: ∀ *w d*. *d* ∉ *c1*‘ *sentences* ⟶ *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w d* = *f3* **using** *GM* **by** *simp*
have *B*: ∀ *w d*. *d* ∉ *c1*‘ *sentences* ⟶ *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w d* = *f3* **using** *GM* **by** *simp*
from *A B* **have** *Res2*: ∀ *w d*. *d* ∉ *c1*‘ *sentences* ⟶ *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w d* ≤ *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w d* **by** *simp*

have ∀ *w d*. *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w d* ≤ *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w d*
using *Res1 Res2* **by** *smt*
then show *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) ≤ *fst* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*))
using *le-funI* **by** *smt*
qed

lemma *jumptp-μ-monot2*: ‹ *ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*
⟹ *g1* ≤ *g2* ⟹ *h1* ≤ *h2*
⟹ *snd* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) ≤

snd (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) ›
proof−
assume *GM*: *ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*
assume *H1*: *g1* ≤ *g2*
assume *H2*: *h1* ≤ *h2*

from *GM H1 H2* **have** *1*: ∀ *w v1 v2*. *v1* ∈ *c1*‘*sentences* ∧ *v2* ∈ *range c2* ⟶
snd (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w v1 v2* ≤
snd (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w v1 v2*
using *pVal-μ-monot* **by** (*smt jumptpμ.simps snd-conv*)

from *GM* **have** *2*: ∀ *w v1 v2*. *v1* ∉ *c1*‘*sentences* ∨ *v2* ∉ *range c2* ⟶
snd (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)) *w v1 v2* ≤
snd (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)) *w v1 v2* **by** *simp*
from *1 2* **show** *?thesis* **using** *le-funI* **by** *smt*
qed

lemma *μtp-fixed-point-prop*:
‹*ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* ⟹
(∃ *g h*. *jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g*,*h*) = (*g*,*h*))›
proof−
assume *GM*: *ground-tpmod f3 t3* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*
then have *H*: *c1*‘ *sentences* ⊆ *D* **by** *simp*
let *?U1* = (*UNIV* :: (′*w* ⇒ ′*v* ⇒ *bool3*) *set*)
have *1*: *ccpo ?U1*
using *function-space-ccpo-bool3 function-space-ccpo-full* **by** *auto*

let *?U2* = (*UNIV* :: (′*w* ⇒ ′*v* ⇒ ′*v* ⇒ *bool3*) *set*)
have *2*: *ccpo ?U2*
using *function-space-ccpo-bool3 function-space-ccpo-full* **by** *metis*

let *?U* = *?U1* × *?U2*
from *1 2* **have** *3*: *ccpo ?U* **using** *product-ccpo* **by** *metis*

have ∀ *g1 g2 h1 h2*.
g1 ≤ *g2* ⟶ *h1* ≤ *h2* ⟶ *jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g1*,*h1*)
≤ *jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2* (*g2*,*h2*)
using *jumptp-μ-monot jumptp-μ-monot2 GM* **by** (*metis less-eq-prod-def*)
then have *4*: *monot* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*) **by** *simp*
have *5*: (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*) ‘ *?U* ⊆ *?U* **by** *simp*
from *3 4 5 VisserFixp* **have** *ccpo* (*FixPs ?U* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*))
by *blast*
from *this ccpo-least-element* **have** ∃ *g h*. (*g*,*h*) ∈ (*FixPs ?U* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*)) **by** *fast*
then obtain *g h* **where** *Hg*: ‹(*g*,*h*) ∈ (*FixPs ?U* (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*))› **by** *auto*
show ∃ *g h*. (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*) (*g*,*h*) = (*g*,*h*) **proof**
show ∃ *h*. (*jumptpμ* (*m*, *D*, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) *G H c1 c2*) (*g*,*h*) = (*g*,*h*) **proof**

show *(jumptpμ (m, D, 𝔈w, 𝔉w, ℜw) G H c1 c2) (g, h) = (g, h)* **using** *FixPs-def Hg* **by** *blast*
qed
qed
qed

function *jumptpν*
where
jumptpν (m, D, 𝔈w, 𝔉w, ℜw) G H c1 c2 (g,h) =
(if (ground-tpmod f4 t4 (m, D, 𝔈w, 𝔉w, ℜw) G H c1 c2) then
((λ w A. if (A ∈ c1‘ sentences) then
value-fm ν (λ x. undefined) (modpplus (m, D, 𝔈w, 𝔉w, ℜw) G H g h w) (inv c1 A)
else f4),
(λ w v1 v2. if (v1 ∈ c1‘sentences ∧ v2 ∈ range c2) then
pVal-ν (m, D, 𝔈w, 𝔉w, ℜw) G H g h w (inv c1 v1) (inv c2 v2)
else f4))
else undefined)
apply *auto[1]*
by *blast*
termination by *lexicographic-order*

lemma *jumptp-ν-monot*: ‹ *ground-tpmod f4 t4 (m, D, 𝔈w, 𝔉w, ℜw) G H c1 c2*
⟹ g1 ≤ g2 ⟹ h1 ≤ h2
⟹ fst (jumptpν (m, D, 𝔈w, 𝔉w, ℜw) G H c1 c2 (g1,h1)) ≤
fst (jumptpν (m, D, 𝔈w, 𝔉w, ℜw) G H c1 c2 (g2,h2)) ›
proof−
assume *GM*: *ground-tpmod f4 t4 (m, D, 𝔈w, 𝔉w, ℜw) G H c1 c2*
then have *S*: *c1‘ sentences ⊆ D* **by** *simp*
assume *H1*: *g1 ≤ g2*
assume *H2*: *h1 ≤ h2*
have ∀ *w. (ℜw w)*
(G := (λ val-list. (g1 w) (hd val-list)), H := (λ val-list. (h1 w) (hd val-list) (last val-list)))
≤ (ℜw w)
(G := (λ val-list. (g2 w) (hd val-list)), H := (λ val-list. (h2 w) (hd val-list) (last val-list)))
by *(simp add: H1 H2 le-funD le-funI)*

then have ∀ *w. leqMod (modpplus (m, D, 𝔈w, 𝔉w, ℜw) G H g1 h1 w) (modpplus (m, D, 𝔈w, 𝔉w, ℜw) G H g2 h2 w)*
by *simp*
from *this monot-in-ν* **have** *Res*: ⋀ *s A w.*
value-fm ν s (modpplus (m, D, 𝔈w, 𝔉w, ℜw) G H g1 h1 w) A
≤ value-fm ν s (modpplus (m, D, 𝔈w, 𝔉w, ℜw) G H g2 h2 w) A **by** *fastforce*

have ∀ *w.* ∀ *A ∈ sentences.*
fst (jumptpν (m, D, 𝔈w, 𝔉w, ℜw) G H c1 c2 (g1,h1)) w (c1 A) ≤
fst (jumptpν (m, D, 𝔈w, 𝔉w, ℜw) G H c1 c2 (g2,h2)) w (c1 A) **proof**

fix w
show $\forall\ A \in$ *sentences*.
fst (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g1*,*h1*)) w (*c1* A) $\leq$
fst (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g2*,*h2*)) w (*c1* A) **proof**
fix A::($'d$, $'c$, $'e$) *fm*
assume *Asent*: ‹$A \in$ *sentences*›

from *S this* **have** A: *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g1*,*h1*)) w (*c1* A) $=$
value-fm ν (λ x. *undefined*) (*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g1 h1* w) A
using *GM* **by** *simp*

from *S Asent* **have** B: *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g2*,*h2*)) w (*c1* A) $=$
value-fm ν (λ x. *undefined*) (*modpplus* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *g2 h2* w) A
using *GM* **by** *simp*

show *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g1*,*h1*)) w (*c1* A) $\leq$
fst (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g2*,*h2*)) w (*c1* A)
using A B *Res* **by** *simp*
qed
qed
from *this* **have** *Res1*: $\forall\ w.\ \forall\ d \in$ *c1*‘ *sentences*.
fst (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g1*,*h1*)) w d $\leq$ *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g2*,*h2*)) w d **using** *GM* **by** *simp*

have A: $\forall\ w\ d.\ d \notin$ *c1*‘ *sentences* $\longrightarrow$ *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g1*,*h1*)) w d $=$ *f4* **using** *GM* **by** *simp*
have B: $\forall\ w\ d.\ d \notin$ *c1*‘ *sentences* $\longrightarrow$ *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g2*,*h2*)) w d $=$ *f4* **using** *GM* **by** *simp*
from A B **have** *Res2*: $\forall\ w\ d.\ d \notin$ *c1*‘ *sentences* $\longrightarrow$ *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g1*,*h1*)) w d $\leq$ *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g2*,*h2*)) w d **by** *simp*

have $\forall\ w\ d.$ *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g1*,*h1*)) w d $\leq$ *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g2*,*h2*)) w d
using *Res1 Res2* **by** *smt*
then show *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g1*,*h1*)) $\leq$ *fst* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g2*,*h2*))
using *le-funI* **by** *smt*
qed

lemma *jumptp-*ν*-monot2*: ‹ *ground-tpmod f4 t4* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2*
$\Longrightarrow$ *g1* $\leq$ *g2* $\Longrightarrow$ *h1* $\leq$ *h2*
$\Longrightarrow$ *snd* (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g1*,*h1*)) $\leq$
snd (*jumptp*ν (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2* (*g2*,*h2*)) ›
proof −
assume *GM*: *ground-tpmod f4 t4* (m, D, $\mathfrak{C}w$, $\mathfrak{F}w$, $\mathfrak{R}w$) G H *c1 c2*
assume *H1*: *g1* $\leq$ *g2*

assume *H2*: *h1* ≤ *h2*

from *GM H1 H2* **have** *1*: ∀ *w v1 v2*. *v1* ∈ *c1‘sentences* ∧ *v2* ∈ *range c2* ⟶
snd (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g1*,*h1*)) *w v1 v2* ≤
snd (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g2*,*h2*)) *w v1 v2*
using *pVal-ν-monot* **by** (*smt jumptpν.simps snd-conv*)

from *GM* **have** *2*: ∀ *w v1 v2*. *v1* ∉ *c1‘sentences* ∨ *v2* ∉ *range c2* ⟶
snd (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g1*,*h1*)) *w v1 v2* ≤
snd (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g2*,*h2*)) *w v1 v2* **by** *simp*
from *1 2* **show** *?thesis* **using** *le-funI* **by** *smt*
qed

lemma *νtp-fixed-point-prop*:
‹*ground-tpmod f4 t4* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* ⟹
(∃ *g h*. *jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g*,*h*) = (*g*,*h*))›
proof−
assume *GM*: *ground-tpmod f4 t4* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*
then have *H*: *c1‘ sentences* ⊆ *D* **by** *simp*
let *?U1* = (*UNIV* :: (*'w* ⇒ *'v* ⇒ *bool4*) *set*)
have *1*: *ccpo ?U1*
using *function-space-ccpo-bool4 function-space-ccpo-full* **by** *auto*

let *?U2* = (*UNIV* :: (*'w* ⇒ *'v* ⇒ *'v* ⇒ *bool4*) *set*)
have *2*: *ccpo ?U2*
using *function-space-ccpo-bool4 function-space-ccpo-full* **by** *metis*

let *?U* = *?U1* × *?U2*
from *1 2* **have** *3*: *ccpo ?U* **using** *product-ccpo* **by** *metis*

have ∀ *g1 g2 h1 h2*.
g1 ≤ *g2* ⟶ *h1* ≤ *h2* ⟶ *jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g1*,*h1*)
≤ *jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2* (*g2*,*h2*)
using *jumptp-ν-monot jumptp-ν-monot2 GM* **by** (*metis less-eq-prod-def*)
then have *4*: *monot* (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*) **by** *simp*
have *5*: (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*) ‘ *?U* ⊆ *?U* **by** *simp*
from *3 4 5 VisserFixp* **have** *ccpo* (*FixPs ?U* (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*))
by *blast*
from *this ccpo-least-element* **have** ∃ *g h*. (*g*,*h*) ∈ (*FixPs ?U* (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*)) **by** *fast*
then obtain *g h* **where** *Hg*: ‹(*g*,*h*) ∈ (*FixPs ?U* (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*))› **by** *auto*
show ∃ *g h*. (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*) (*g*,*h*) = (*g*,*h*) **proof**
show ∃ *h*. (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*) (*g*,*h*) = (*g*,*h*) **proof**
show (*jumptpν* (*m*, *D*, 𝔆*w*, 𝔉*w*, ℜ*w*) *G H c1 c2*) (*g*, *h*) = (*g*, *h*) **using** *FixPs-def Hg* **by** *blast*
qed
qed

qed

end