A Verified Reduction Algorithm from MLSSmf to MLSS

Yiran Duan, Lukas Stevens

February 6, 2026

Abstract

Multi-level syllogistic with monotone functions (**MLSSmf**) is a sublanguage of set theory introduced by Cantone et al. [1], involving set-to-set functions and their monotonicity, additivity, and multiplicativity. It is an extension of *multi-level syllogistic with singleton* (**MLSS**), which involves the predicates membership, set equality, set inclusion, and the operators union, intersection, set difference, and singleton.

In this work we formalize the reduction algorithm from **MLSSmf** to **MLSS**, and verify the correctness proof originally presented by Cantone et al. [1]. Combined with the verified decision procedure for **MLSS** formalized by Stevens [2], this yields a verified decision procedure for **MLSSmf**.

theory *MLSSmf-to-MLSS-Complexity*
imports *MLSSmf-to-MLSS*
begin

definition $size_m :: ('v, 'f)$ *MLSSmf-clause* $\Rightarrow$ *nat* **where**
$size_m\ \mathcal{C} \equiv card\ (set\ \mathcal{C})$

lemma (**in** *normalized-MLSSmf-clause*) *card-V-upper-bound*:
card $V \leq 3 * size_m\ \mathcal{C}$
unfolding *V-def*
using *norm-*$\mathcal{C}$
proof (*induction* $\mathcal{C}$)
case *1*
then show *?case* **by** *simp*
next
case (*2 ls l*)
from ‹*norm-literal l*› **have** *card* $(vars_m\ l) \leq 3$
by (*cases l rule*: *norm-literal.cases*) (*auto simp*: *card-insert-if*)
with *2* **show** *?case*
proof (*cases* $l \in set\ ls$)
case *True*
then have $vars_m\ l \subseteq vars_m\ ls$ **by** *blast*
moreover
have $vars_m\ (l\ \#\ ls) = vars_m\ l \cup vars_m\ ls$ **by** *auto*
ultimately
have $vars_m\ (l\ \#\ ls) = vars_m\ ls$ **by** *blast*
then have $card\ (vars_m\ (l\ \#\ ls)) = card\ (vars_m\ ls)$ **by** *argo*
moreover
from *True* **have** $size_m\ (l\ \#\ ls) = size_m\ ls$
unfolding $size_m$*-def*
by (*simp add*: *insert-absorb*)
ultimately
show *?thesis* **using** *2.IH* **by** *argo*
next
case *False*
have $vars_m\ (l\ \#\ ls) = vars_m\ l \cup vars_m\ ls$ **by** *auto*
then have $card\ (vars_m\ (l\ \#\ ls)) \leq card\ (vars_m\ l) + card\ (vars_m\ ls)$
by (*simp add*: *card-Un-le*)
with ‹$card\ (vars_m\ l) \leq 3$› *2.IH*
have $card\ (vars_m\ (l\ \#\ ls)) \leq 3 * (Suc\ (size_m\ ls))$
by *simp*
moreover
from *False* **have** $size_m\ (l\ \#\ ls) = Suc\ (size_m\ ls)$
unfolding $size_m$*-def* **by** *simp*
ultimately
show *?thesis* **by** *argo*
qed
qed

lemma (**in** *normalized-MLSSmf-clause*) *card-F-upper-bound*:
card F ≤ *2* ∗ *size*$_m$ $\mathcal{C}$
unfolding *F-def*
using *norm-*$\mathcal{C}$
proof (*induction* $\mathcal{C}$)
case *1*
then show *?case* **by** *simp*
next
case (*2 ls l*)
from ‹*norm-literal l*› **have** *card* (*funs*$_m$ *l*) ≤ *2*
by (*cases l rule*: *norm-literal.cases*) (*auto simp*: *card-insert-if*)
with *2* **show** *?case*
proof (*cases l* ∈ *set ls*)
case *True*
then have *funs*$_m$ *l* ⊆ *funs*$_m$ *ls* **by** *blast*
moreover
have *funs*$_m$ (*l* # *ls*) = *funs*$_m$ *l* ∪ *funs*$_m$ *ls* **by** *auto*
ultimately
have *funs*$_m$ (*l* # *ls*) = *funs*$_m$ *ls* **by** *blast*
then have *card* (*funs*$_m$ (*l* # *ls*)) = *card* (*funs*$_m$ *ls*) **by** *argo*
moreover
from *True* **have** *size*$_m$ (*l* # *ls*) = *size*$_m$ *ls*
unfolding *size*$_m$*-def*
by (*simp add*: *insert-absorb*)
ultimately
show *?thesis* **using** *2.IH* **by** *argo*
next
case *False*
have *funs*$_m$ (*l* # *ls*) = *funs*$_m$ *l* ∪ *funs*$_m$ *ls* **by** *auto*
then have *card* (*funs*$_m$ (*l* # *ls*)) ≤ *card* (*funs*$_m$ *l*) + *card* (*funs*$_m$ *ls*)
by (*simp add*: *card-Un-le*)
with ‹*card* (*funs*$_m$ *l*) ≤ *2*› *2.IH*
have *card* (*funs*$_m$ (*l* # *ls*)) ≤ *2* ∗ (*Suc* (*size*$_m$ *ls*))
by *simp*
moreover
from *False* **have** *size*$_m$ (*l* # *ls*) = *Suc* (*size*$_m$ *ls*)
unfolding *size*$_m$*-def* **by** *simp*
ultimately
show *?thesis* **by** *argo*
qed
qed

lemma (**in** *normalized-MLSSmf-clause*) *size-restriction-on-InterOfVars*:
card (*restriction-on-InterOfVars vs*) ≤ *2* ∗ *length vs*
proof (*induction vs rule*: *restriction-on-InterOfVars.induct*)
case (*3 x v vs*)
have *length zs* > *length ys* ⟹ *InterOfVarsAux zs* ∉ ⋃ (*vars* ‘ *restriction-on-InterOfVars ys*)
for *y ys zs*

by (*induction ys rule*: *restriction-on-InterOfVars.induct*) *auto*
then have *InterOfVarsAux* (*x # v # vs*) ∉ ⋃ (*vars* ‘ *restriction-on-InterOfVars* (*v # vs*))
by *force*
then have *Var* (*InterOfVarsAux* (*x # v # vs*)) =$_s$ *Var* (*Solo x*) −$_s$ *Var* (*InterOfVars* (*v # vs*)) ∉ *restriction-on-InterOfVars* (*v # vs*)
Var (*InterOfVars* (*x # v # vs*)) =$_s$ *Var* (*Solo x*) −$_s$ *Var* (*InterOfVarsAux* (*x # v # vs*)) ∉ *restriction-on-InterOfVars* (*v # vs*)
by *auto*
then have *card* (*restriction-on-InterOfVars* (*x # v # vs*)) = *Suc* (*Suc* (*card* (*restriction-on-InterOfVars* (*v # vs*))))
using *restriction-on-InterOfVar-finite* **by** *force*
with *3.IH* **show** *?case* **by** *simp*
qed *simp+*

lemma (**in** *normalized-MLSSmf-clause*) *size-restriction-on-UnionOfVars*:
card (*restriction-on-UnionOfVars vs*) ≤ *Suc* (*length vs*)
apply (*induction vs rule*: *restriction-on-UnionOfVars.induct*)
apply *simp*
by (*simp add*: *card-insert-if restriction-on-UnionOfVar-finite*)

theorem (**in** *normalized-MLSSmf-clause*) *size-introduce-v*:
card introduce-v ≤ (*3* ∗ *card V* + *2*) ∗ (*2* ^ *card V*)
proof −
have *card* (*restriction-on-v* ‘ *P*$^+$ *V*) ≤ *card* (*P*$^+$ *V*)
using *P-plus-finite card-image-le* **by** *blast*
then have *1*: *card* (*restriction-on-v* ‘ *P*$^+$ *V*) ≤ *card* (*Pow V*)
by *simp*

have *card* ((*restriction-on-InterOfVars* ∘ *var-set-to-list*) *α*) ≤ *2* ∗ *card V* **for** *α*
proof −
have *length* (*var-set-to-list α*) ≤ *length V-list* **by** *simp*
then have *length* (*var-set-to-list α*) ≤ *card V*
unfolding *V-list-def*
by (*metis V-list-def distinct-V-list distinct-card set-V-list*)
with *size-restriction-on-InterOfVars*[*of var-set-to-list α*]
have *card* (*restriction-on-InterOfVars* (*var-set-to-list α*)) ≤ *2* ∗ *card V*
by *linarith*
then show *?thesis* **by** *fastforce*
qed
then have (∑ *α*∈*P*$^+$ *V*. *card* ((*restriction-on-InterOfVars* ∘ *var-set-to-list*) *α*)) ≤ *2* ∗ *card V* ∗ *card* (*P*$^+$ *V*)
by (*smt* (*verit*) *card-eq-sum nat-mult-1-right sum-distrib-left sum-mono*)
moreover
from *card-UN-le*[**where** *?I* = *P*$^+$ *V* **and** *?A* = *restriction-on-InterOfVars* ∘ *var-set-to-list*]
have *card* (⋃ ((*restriction-on-InterOfVars* ∘ *var-set-to-list*) ‘ *P*$^+$ *V*)) ≤
(∑ *α*∈*P*$^+$ *V*. *card* ((*restriction-on-InterOfVars* ∘ *var-set-to-list*) *α*))
using *P-plus-finite finite-V* **by** *blast*

ultimately
have *card* ($\bigcup$ ((*restriction-on-InterOfVars* $\circ$ *var-set-to-list*) ‘ P^+ *V*)) $\leq$ *2* $*$ *card V* $*$ *card* (P^+ *V*)
by *linarith*
also have ... $\leq$ *2* $*$ *card V* $*$ *card* (*Pow V*) **by** *simp*
finally have *2*: *card* ($\bigcup$ ((*restriction-on-InterOfVars* $\circ$ *var-set-to-list*) ‘ P^+ *V*)) $\leq$ *2* $*$ *card V* $*$ *card* (*Pow V*)
by *blast*

have *card* ((*restriction-on-UnionOfVars* $\circ$ *var-set-to-list*) α) $\leq$ *Suc* (*card V*) **for** α
proof −
have *length* (*var-set-to-list* α) $\leq$ *length V-list* **by** *simp*
then have *length* (*var-set-to-list* α) $\leq$ *card V*
unfolding *V-list-def*
by (*metis V-list-def distinct-V-list distinct-card set-V-list*)
with *size-restriction-on-UnionOfVars*[*of var-set-to-list* α]
have *card* (*restriction-on-UnionOfVars* (*var-set-to-list* α)) $\leq$ *Suc* (*card V*)
by *linarith*
then show *?thesis* **by** *fastforce*
qed
then have ($\sum \alpha \in$ *Pow V*. *card* ((*restriction-on-UnionOfVars* $\circ$ *var-set-to-list*) α)) $\leq$ *Suc* (*card V*) $*$ *card* (*Pow V*)
by (*smt* (*verit*) *card-eq-sum nat-mult-1-right sum-distrib-left sum-mono*)
moreover
from *card-UN-le*[**where** *?I* = *Pow V* **and** *?A* = *restriction-on-UnionOfVars* $\circ$ *var-set-to-list*]
have *card* ($\bigcup$ ((*restriction-on-UnionOfVars* $\circ$ *var-set-to-list*) ‘ *Pow V*)) $\leq$
($\sum \alpha \in$ *Pow V*. *card* ((*restriction-on-UnionOfVars* $\circ$ *var-set-to-list*) α))
using *finite-V* **by** *blast*
ultimately
have *3*: *card* ($\bigcup$ ((*restriction-on-UnionOfVars* $\circ$ *var-set-to-list*) ‘ *Pow V*)) $\leq$ *Suc* (*card V*) $*$ *card* (*Pow V*)
by *linarith*

let *?atoms* = *restriction-on-v* ‘ P^+ *V* $\cup$
$\bigcup$ ((*restriction-on-InterOfVars* $\circ$ *var-set-to-list*) ‘ P^+ *V*) $\cup$
$\bigcup$ ((*restriction-on-UnionOfVars* $\circ$ *var-set-to-list*) ‘ *Pow V*)
from *restriction-on-InterOfVar-finite restriction-on-UnionOfVar-finite*
have *finite ?atoms* **using** *finite-V* **by** *auto*
then have *card introduce-v* $\leq$ *card ?atoms*
unfolding *introduce-v-def*
using *card-image-le* **by** *meson*
also have ... $\leq$ *card* (*restriction-on-v* ‘ P^+ *V*) +
card ($\bigcup$ ((*restriction-on-InterOfVars* $\circ$ *var-set-to-list*) ‘ P^+ *V*)) +
card ($\bigcup$ ((*restriction-on-UnionOfVars* $\circ$ *var-set-to-list*) ‘ *Pow V*))
using *finite-V* **by** (*auto intro*!: *order.trans*[*OF card-Un-le*])
also have ... $\leq$ *card* (*Pow V*) +
card ($\bigcup$ ((*restriction-on-InterOfVars* $\circ$ *var-set-to-list*) ‘ P^+ *V*)) +

```
        card (⋃ ((restriction-on-UnionOfVars ∘ var-set-to-list) ` Pow V))
    using 1 by linarith
  also have ... ≤ card (Pow V) + 2 * card V * card (Pow V) +
        card (⋃ ((restriction-on-UnionOfVars ∘ var-set-to-list) ` Pow V))
    using 2 by linarith
  also have ... ≤ card (Pow V) + 2 * card V * card (Pow V) + Suc (card V) *
card (Pow V)
    using 3 by linarith
  also have ... = (1 + 2 * card V + Suc (card V)) * card (Pow V)
    by algebra
  also have ... = (3 * card V + 2) * card (Pow V)
    by simp
  also have ... = (3 * card V + 2) * (2 ^ card V)
    using card-Pow finite-V by fastforce
  finally show ?thesis .
qed

lemma (in normalized-MLSSmf-clause) size-restriction-on-UnionOfVennRegions:
  card (restriction-on-UnionOfVennRegions αs) ≤ Suc (length αs)
  apply (induction αs rule: restriction-on-UnionOfVennRegions.induct)
   apply simp+
   by (metis add-mono-thms-linordered-semiring(2) card.infinite card-insert-if fi-
nite-insert le-SucI plus-1-eq-Suc)

lemma (in normalized-MLSSmf-clause) length-all-V-set-lists:
  length all-V-set-lists = 2 ^ card (P⁺ V)
  unfolding all-V-set-lists-def
  using length-subseqs set-V-set-list distinct-V-set-list distinct-card
  by force

lemma (in normalized-MLSSmf-clause) length-F-list:
  length F-list = card F
  unfolding F-list-def F-def
  by (auto simp add: length-remdups-card-conv)

lemma (in normalized-MLSSmf-clause) size-introduce-UnionOfVennRegions:
  card introduce-UnionOfVennRegions ≤ Suc (2 ^ card V) * 2 ^ 2 ^ card V
proof −
  have 1: card (restriction-on-UnionOfVennRegions αs) ≤ Suc (2 ^ card V)
    if αs ∈ set all-V-set-lists for αs
  proof −
    from that have length αs ≤ length V-set-list
      unfolding all-V-set-lists-def
      using length-subseq-le by blast
    then have length αs ≤ card (P⁺ V)
      by (metis distinct-V-set-list distinct-card set-V-set-list)
    then have length αs ≤ 2 ^ card V
      using card-Pow finite-V by fastforce
    with size-restriction-on-UnionOfVennRegions[of αs]
```

```
    have card (restriction-on-UnionOfVennRegions αs) ≤ Suc (2 ^ card V)
      by linarith
    then show ?thesis by fastforce
  qed

  from length-all-V-set-lists have card (set all-V-set-lists) = 2 ^ card (P⁺ V)
    using distinct-card distinct-all-V-set-lists by metis
  also have ... ≤ 2 ^ card (Pow V) by auto
  also have ... = 2 ^ 2 ^ card V
    using finite-V by (simp add: card-Pow)
  finally have 2: card (set all-V-set-lists) ≤ 2 ^ 2 ^ card V.

  let ?atoms = ⋃ (restriction-on-UnionOfVennRegions ` set all-V-set-lists)
  from AT-inj have inj-on AT ?atoms
    using inj-on-def by force
  from 1 have (∑ αs∈set all-V-set-lists. card (restriction-on-UnionOfVennRegions
αs)) ≤
    Suc (2 ^ card V) * (card (set all-V-set-lists))
    using Sum-le-times[where ?s = set all-V-set-lists
                        and ?f = λαs. card (restriction-on-UnionOfVennRegions αs)]
    by blast
  with 2 have (∑ αs∈set all-V-set-lists. card (restriction-on-UnionOfVennRegions
αs)) ≤
    Suc (2 ^ card V) * 2 ^ 2 ^ card V
    by (meson Suc-mult-le-cancel1 le-trans)
  moreover
  from card-UN-le[where ?I = set all-V-set-lists and ?A = restriction-on-UnionOfVennRegions]
  have card ?atoms ≤ (∑ αs∈set all-V-set-lists. card (restriction-on-UnionOfVennRegions
αs))
    by blast
  ultimately
  have card ?atoms ≤ Suc (2 ^ card V) * 2 ^ 2 ^ card V
    by linarith
  moreover
  from introduce-UnionOfVennRegions-normalized
  have finite introduce-UnionOfVennRegions
    unfolding normalized-MLSS-clause-def by blast
  then have finite ?atoms
    using finite-image-iff ‹inj-on AT ?atoms›
    unfolding introduce-UnionOfVennRegions-def by blast
  ultimately
  show ?thesis
    unfolding introduce-UnionOfVennRegions-def
    using card-image[where ?f = AT and ?A = ?atoms]
    using ‹inj-on AT ?atoms›
    by presburger
qed

lemma (in normalized-MLSSmf-clause) length-choices-from-lists:
```

```
  ∀ choice ∈ set (choices-from-lists xss). length choice = length xss
  by (induction xss) auto

lemma (in normalized-MLSSmf-clause) size-introduce-w:
  ∀ clause ∈ introduce-w. card clause ≤ 2 ^ (2 * 2 ^ card V) * card F
proof
  let ?xss = map (λ(l, m, f). restriction-on-FunOfUnionOfVennRegions l m f)
                 (List.product all-V-set-lists (List.product all-V-set-lists F-list))
  fix clause assume clause ∈ introduce-w
  then obtain choice where choice: choice ∈ set (choices-from-lists ?xss) clause
= set choice
    unfolding introduce-w-def by auto
  then have card clause ≤ length choice
    using card-length by blast
  also have length choice = length ?xss
    using choice length-choices-from-lists by blast
  also have ... = length (List.product all-V-set-lists (List.product all-V-set-lists
F-list))
    by simp
  also have ... = length all-V-set-lists * length all-V-set-lists * length F-list
    using length-product by auto
  also have ... = 2 ^ card (P⁺ V) * 2 ^ card (P⁺ V) * card F
    using length-all-V-set-lists length-F-list by presburger
  also have ... = 2 ^ (2 * (card (P⁺ V))) * card F
    by (simp add: mult-2 power-add)
  also have ... ≤ 2 ^ (2 * (card (Pow V))) * card F
    by simp
  also have ... = 2 ^ (2 * 2 ^ card V) * card F
    using card-Pow by auto
  finally show card clause ≤ 2 ^ (2 * 2 ^ card V) * card F .
qed

lemma (in normalized-MLSSmf-clause) card-P-P-V-ge-1:
  card (Pow (P⁺ V) × Pow (P⁺ V)) ≥ 1
proof −
  have Pow (P⁺ V) ≠ {} by blast
  then have Pow (P⁺ V) × Pow (P⁺ V) ≠ {} by blast
  moreover
  from finite-V P-plus-finite have finite (Pow (P⁺ V)) by blast
  then have finite (Pow (P⁺ V) × Pow (P⁺ V)) by blast
  ultimately
  have card (Pow (P⁺ V) × Pow (P⁺ V)) > 0 by auto
  then show ?thesis by linarith
qed

lemma (in normalized-MLSSmf-clause) size-reduce-norm-literal:
  assumes norm-literal lt
    shows card (reduce-literal lt) ≤ 2 * card (Pow (P⁺ V) × Pow (P⁺ V))
  using assms
```

proof (*cases lt rule*: *norm-literal.cases*)
case (*inc f*)
let *?l* = $\lambda(l, m)$. *AT* (*Var* w_{fm} =$_s$ *Var* w_{fm} ⊔$_s$ *Var* w_{fl})
from *inc* **have** *reduce-literal lt* ⊆ *?l* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by *force*
then have *card* (*reduce-literal lt*) ≤ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by (*meson finite-SigmaI finite-V pow-of-p-Plus-finite surj-card-le*)
also have ... ≤ *2* ∗ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) **by** *linarith*
finally show *?thesis* **.**
next
case (*dec f*)
let *?l* = $\lambda(l, m)$. *AT* (*Var* w_{fl} =$_s$ *Var* w_{fl} ⊔$_s$ *Var* w_{fm})
from *dec* **have** *reduce-literal lt* ⊆ *?l* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by *force*
then have *card* (*reduce-literal lt*) ≤ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by (*meson finite-SigmaI finite-V pow-of-p-Plus-finite surj-card-le*)
also have ... ≤ *2* ∗ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) **by** *linarith*
finally show *?thesis* **.**
next
case (*add f*)
let *?l* = $\lambda(l, m)$. *AT* (*Var* $w_{fl \cup m}$ =$_s$ *Var* w_{fl} ⊔$_s$ *Var* w_{fm})
from *add* **have** *reduce-literal lt* ⊆ *?l* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by *force*
then have *card* (*reduce-literal lt*) ≤ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by (*meson finite-SigmaI finite-V pow-of-p-Plus-finite surj-card-le*)
also have ... ≤ *2* ∗ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) **by** *linarith*
finally show *?thesis* **.**
next
case (*mul f*)
let *?l1* = $\lambda(l, m)$. *AT* (*Var* (*InterOfWAux f l m*) =$_s$ *Var* w_{fl} −$_s$ *Var* w_{fm})
let *?l2* = $\lambda(l, m)$. *AT* (*Var* $w_{fl \cap m}$ =$_s$ *Var* w_{fl} −$_s$ *Var* (*InterOfWAux f l m*))
from *mul* **have** *reduce-literal lt* ⊆ *?l1* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) ∪ *?l2* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by *force*
moreover
have *?l1* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) ∩ *?l2* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) = {}
by *fastforce*
moreover
from *finite-V P-plus-finite* **have** *finite* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by *auto*
then have *finite* (*?l1* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))) *finite* (*?l2* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)))
by *blast+*
ultimately
have *card* (*reduce-literal lt*) ≤ *card* (*?l1* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))) + *card* (*?l2* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)))
using *card-Un-disjoint*[**where** *?A* = *?l1* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) **and** *?B* = *?l2* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))]

using *card-mono*[**where** *?A* = *reduce-literal lt* **and** *?B* = *?l1* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) ∪ *?l2* ‘ (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))]
by *fastforce*
also have ... ≤ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) + *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
using *card-image-le*[**where** *?A* = *Pow* (P^+ *V*) × *Pow* (P^+ *V*)]
using ‹*finite* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))› *add-mono* **by** *blast*
also have ... = *2* ∗ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*)) **by** *linarith*
finally show *?thesis* **.**
next
case (*le f g*)
let *?l* = λ*l*. *AT* (*Var* w_{gl} =$_s$ *Var* w_{gl} ⊔$_s$ *Var* w_{fl})
from *le* **have** *reduce-literal lt* ⊆ *?l* ‘ *Pow* (P^+ *V*)
by *force*
then have *card* (*reduce-literal lt*) ≤ *card* (*Pow* (P^+ *V*))
by (*simp add*: *finite-V surj-card-le*)
also have ... ≤ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by (*simp add*: *finite-V surj-card-le*)
also have ... ≤ *2* ∗ *card* (*Pow* (P^+ *V*) × *Pow* (P^+ *V*))
by *linarith*
finally show *?thesis* **.**
next
case (*eq x y*)
then have *card* (*reduce-literal lt*) = *1* **by** *simp*
with *card-P-P-V-ge-1* **show** *?thesis* **by** *linarith*
next
case (*eq-empty x n*)
then have *card* (*reduce-literal lt*) = *1* **by** *simp*
with *card-P-P-V-ge-1* **show** *?thesis* **by** *linarith*
next
case (*neq x y*)
then have *card* (*reduce-literal lt*) = *1* **by** *simp*
with *card-P-P-V-ge-1* **show** *?thesis* **by** *linarith*
next
case (*union x y z*)
then have *card* (*reduce-literal lt*) = *1* **by** *simp*
with *card-P-P-V-ge-1* **show** *?thesis* **by** *linarith*
next
case (*diff x y z*)
then have *card* (*reduce-literal lt*) = *1* **by** *simp*
with *card-P-P-V-ge-1* **show** *?thesis* **by** *linarith*
next
case (*single x y*)
then have *card* (*reduce-literal lt*) = *1* **by** *simp*
with *card-P-P-V-ge-1* **show** *?thesis* **by** *linarith*
next
case (*app x f y*)
then have *card* (*reduce-literal lt*) = *1* **by** *simp*
with *card-P-P-V-ge-1* **show** *?thesis* **by** *linarith*

qed

lemma (**in** *normalized-MLSSmf-clause*) *size-reduce-clause*:
card reduce-clause $\leq 2 \hat{} (Suc\ (2 * 2 \hat{} card\ V)) * size_m\ \mathcal{C}$
proof −
have *card* $(P^+\ V) \leq 2 \hat{} card\ V$
using *card-Pow*[*of V*] *finite-V* **by** *simp*
from *card-UN-le*
have *card reduce-clause* $\leq (\sum lt \in set\ \mathcal{C}.\ card\ (reduce\text{-}literal\ lt))$
using *reduce-clause-finite*
unfolding *reduce-clause-def*
by *blast*
also have ... $\leq 2 * card\ (Pow\ (P^+\ V) \times Pow\ (P^+\ V)) * card\ (set\ \mathcal{C})$
using *size-reduce-norm-literal norm-C literal-in-norm-clause-is-norm*
using *Sum-le-times*[**where** *?s* = *set* $\mathcal{C}$ **and** *?f* = $\lambda lt.$ *card* (*reduce-literal lt*)
and *?n* = $2 * card\ (Pow\ (P^+\ V) \times Pow\ (P^+\ V))$]
by *blast*
also have ... $= 2 * card\ (Pow\ (P^+\ V)) * card\ (Pow\ (P^+\ V)) * card\ (set\ \mathcal{C})$
using *card-cartesian-product* **by** *auto*
also have ... $= 2 * 2 \hat{} (card\ (P^+\ V)) * 2 \hat{} (card\ (P^+\ V)) * card\ (set\ \mathcal{C})$
using *card-Pow*[*of* P^+ *V*] *finite-V P-plus-finite* **by** *fastforce*
also have ... $\leq 2 * 2 \hat{} (2 \hat{} card\ V) * 2 \hat{} (2 \hat{} card\ V) * card\ (set\ \mathcal{C})$
using ‹*card* $(P^+\ V) \leq 2 \hat{} card\ V$›
using *power-increasing-iff*[**where** *?b* = 2 **and** *?x* = *card* $(P^+\ V)$ **and** *?y* = 2
$\hat{}$ *card V*]
by (*simp add*: *mult-le-mono*)
also have ... $= 2 \hat{} (Suc\ (2 * 2 \hat{} card\ V)) * card\ (set\ \mathcal{C})$
by (*simp add*: *power2-eq-square power-even-eq*)
also have ... $= 2 \hat{} (Suc\ (2 * 2 \hat{} card\ V)) * size_m\ \mathcal{C}$
unfolding $size_m$*-def* **by** *blast*
finally show *?thesis* **.**
qed

theorem (**in** *normalized-MLSSmf-clause*) *size-reduced-dnf*:
$\forall\ clause \in reduced\text{-}dnf.\ card\ clause \leq$
$2 \hat{} (2 * 2 \hat{} (3 * size_m\ \mathcal{C})) * (2 * size_m\ \mathcal{C}) +$
$(3 * (3 * size_m\ \mathcal{C}) + 2) * (2 \hat{} (3 * size_m\ \mathcal{C})) +$
$Suc\ (2 \hat{} (3 * size_m\ \mathcal{C})) * 2 \hat{} 2 \hat{} (3 * size_m\ \mathcal{C}) +$
$2 \hat{} (Suc\ (2 * 2 \hat{} (3 * size_m\ \mathcal{C}))) * size_m\ \mathcal{C}$
proof −
let *?upper-bound* $= 2 \hat{} (2 * 2 \hat{} (3 * size_m\ \mathcal{C})) * (2 * size_m\ \mathcal{C}) +$
$(3 * (3 * size_m\ \mathcal{C}) + 2) * (2 \hat{} (3 * size_m\ \mathcal{C})) +$
$Suc\ (2 \hat{} (3 * size_m\ \mathcal{C})) * 2 \hat{} 2 \hat{} (3 * size_m\ \mathcal{C}) +$
$2 \hat{} (Suc\ (2 * 2 \hat{} (3 * size_m\ \mathcal{C}))) * size_m\ \mathcal{C}$
{fix *clause* **assume** *clause* $\in$ *reduced-dnf*
then obtain *fms* **where** *fms* $\in$ *introduce-w*
and *clause*: *clause* = *fms* $\cup$ *introduce-v* $\cup$ *introduce-UnionOfVennRegions*
$\cup$ *reduce-clause*
unfolding *reduced-dnf-def* **by** *blast*

then have *card clause* $\leq$ *card fms* + *card introduce-v* + *card introduce-UnionOfVennRegions* + *card reduce-clause*
by (*auto intro*!: *order.trans*[*OF card-Un-le*])
also have ... $\leq$ $2 \mathbin{\widehat{}} (2 * 2 \mathbin{\widehat{}} \textit{card}\ V) * \textit{card}\ F + \textit{card}$ *introduce-v* + *card introduce-UnionOfVennRegions* + *card reduce-clause*
using *size-introduce-w* ‹*fms* $\in$ *introduce-w*› **by** *fastforce*
also have ... $\leq 2 \mathbin{\widehat{}} (2 * 2 \mathbin{\widehat{}} \textit{card}\ V) * \textit{card}\ F + (3 * \textit{card}\ V + 2) * (2 \mathbin{\widehat{}} \textit{card}$ $V)$ + *card introduce-UnionOfVennRegions* + *card reduce-clause*
using *size-introduce-v* **by** *simp*
also have ... $\leq 2 \mathbin{\widehat{}} (2 * 2 \mathbin{\widehat{}} \textit{card}\ V) * \textit{card}\ F + (3 * \textit{card}\ V + 2) * (2 \mathbin{\widehat{}} \textit{card}$ $V) + \textit{Suc}\ (2 \mathbin{\widehat{}} \textit{card}\ V) * 2 \mathbin{\widehat{}} 2 \mathbin{\widehat{}} \textit{card}\ V$ + *card reduce-clause*
using *size-introduce-UnionOfVennRegions* **by** *simp*
also have ... $\leq 2 \mathbin{\widehat{}} (2 * 2 \mathbin{\widehat{}} \textit{card}\ V) * \textit{card}\ F + (3 * \textit{card}\ V + 2) * (2 \mathbin{\widehat{}} \textit{card}$ $V) + \textit{Suc}\ (2 \mathbin{\widehat{}} \textit{card}\ V) * 2 \mathbin{\widehat{}} 2 \mathbin{\widehat{}} \textit{card}\ V + 2 \mathbin{\widehat{}} (\textit{Suc}\ (2 * 2 \mathbin{\widehat{}} \textit{card}\ V)) * \textit{size}_m\ \mathcal{C}$
using *size-reduce-clause* **by** *simp*
also have ... $\leq$ *?upper-bound*
using *card-V-upper-bound card-F-upper-bound*
by (*metis Suc-le-mono add-le-mono add-le-mono1 mult-le-mono mult-le-mono1 mult-le-mono2 one-le-numeral power-increasing*)
finally have *card clause* $\leq$ *?upper-bound* **.**
}
then show *?thesis* **by** *blast*
qed

end
theory *MLSSmf-to-MLSS-Soundness*
imports *MLSSmf-to-MLSS MLSSmf-Semantics Proper-Venn-Regions MLSSmf-HF-Extras*
begin

locale *satisfiable-normalized-MLSSmf-clause* =
normalized-MLSSmf-clause $\mathcal{C}$ **for** $\mathcal{C}$ :: ('*v*, '*f*) *MLSSmf-clause* +
fixes M_v :: '*v* $\Rightarrow$ *hf*
and M_f :: '*f* $\Rightarrow$ *hf* $\Rightarrow$ *hf*
assumes *model-for-*$\mathcal{C}$: $I_{cl}\ M_v\ M_f\ \mathcal{C}$
begin

interpretation *proper-Venn-regions* $V\ M_v$
using *finite-V* **by** *unfold-locales*

function $\mathcal{M}$ :: ('*v*, '*f*) *Composite* $\Rightarrow$ *hf* **where**
$\mathcal{M}\ (\textit{Solo}\ x) = M_v\ x$
| $\mathcal{M}\ (v_\alpha) = \textit{proper-Venn-region}\ \alpha$
| $\mathcal{M}\ (\textit{UnionOfVennRegions}\ \textit{xss}) = \bigsqcup \textit{HF}\ ((\mathcal{M} \circ \textit{VennRegion})\ `\ \textit{set xss})$
| $\mathcal{M}\ (w_{fl}) = (M_f\ f)\ (\mathcal{M}\ (\textit{UnionOfVennRegions}\ (\textit{var-set-set-to-var-set-list}\ l)))$
| $\mathcal{M}\ (\textit{UnionOfVars}\ \textit{xs}) = \bigsqcup \textit{HF}\ (M_v\ `\ \textit{set xs})$
| $\mathcal{M}\ (\textit{InterOfVars}\ \textit{xs}) = \bigsqcap \textit{HF}\ (M_v\ `\ \textit{set xs})$
| $\mathcal{M}\ (\textit{MemAux}\ x) = \textit{HF}\ \{M_v\ x\}$
| $\mathcal{M}\ (\textit{InterOfWAux}\ f\ l\ m) = \mathcal{M}\ w_{fl} - \mathcal{M}\ w_{fm}$
| $\mathcal{M}\ (\textit{InterOfVarsAux}\ \textit{xs}) = M_v\ (\textit{hd xs}) - \mathcal{M}\ (\textit{InterOfVars}\ (\textit{tl xs}))$

```
  by pat-completeness auto
termination
  apply (relation measure (λcomp. case comp of
                    InterOfVarsAux - ⇒ Suc 0
                  | UnionOfVennRegions - ⇒ Suc 0
                  | w_-- ⇒ Suc (Suc 0)
                  | InterOfWAux - - - ⇒ Suc (Suc (Suc 0))
                  | - ⇒ 0))
      apply auto
  done

lemma soundness-restriction-on-InterOfVars:
  assumes set xs ∈ P⁺ V
    shows ∀ a ∈ restriction-on-InterOfVars xs. I_sa M a
proof (induction xs rule: restriction-on-InterOfVars.induct)
  case (2 x)
  {fix a assume a ∈ restriction-on-InterOfVars [x]
    then have a = Var (InterOfVars [x]) =_s Var (Solo x) by simp
    then have I_sa M a by (simp add: HInter-singleton)
  }
  then show ?case by blast
next
  case (3 y x xs)
 {fix a assume a ∈ restriction-on-InterOfVars (y # x # xs) − restriction-on-InterOfVars
(x # xs)
    then consider a = Var (InterOfVarsAux (y # x # xs)) =_s Var (Solo y) −_s
Var (InterOfVars (x # xs))
                | a = Var (InterOfVars (y # x # xs)) =_s Var (Solo y) −_s Var
(InterOfVarsAux (y # x # xs))
      by fastforce
    then have I_sa M a
    proof (cases)
      case 1
      then show ?thesis by simp
    next
      case 2
      have ⨅ HF (insert (M_v y) (insert (M_v x) (M_v ‘ set xs))) =
            ⨅ (HF ((insert (M_v x) (M_v ‘ set xs))) ◁ M_v y)
        using HF-insert-hinsert by auto
      also have ... = M_v y ⊓ ⨅ HF (insert (M_v x) (M_v ‘ set xs))
        by (simp add: HF-nonempty)
      also have ... = M_v y − (M_v y − ⨅ HF (insert (M_v x) (M_v ‘ set xs)))
        by blast
      finally show ?thesis using 2 by simp
    qed
  }
  with 3.IH show ?case by blast
qed simp
```

lemma *soundness-restriction-on-UnionOfVars*:
assumes *set xs* ∈ *Pow V*
shows ∀ *a* ∈ *restriction-on-UnionOfVars xs*. I_{sa} $\mathcal{M}$ *a*
proof (*induction xs rule*: *restriction-on-UnionOfVars.induct*)
case *1*
then show *?case* **by** *auto*
next
case (*2 x xs*)
{fix *a* **assume** *a* ∈ *restriction-on-UnionOfVars* (*x* # *xs*) − *restriction-on-UnionOfVars xs*
then have *a*: *a* = *Var* (*UnionOfVars* (*x* # *xs*)) =$_s$ *Var* (*Solo x*) ⊔$_s$ *Var* (*UnionOfVars xs*)
by *fastforce*
have ⨆ *HF* (*insert* (M_v *x*) (M_v ‘ *set xs*)) = ⨆ (*HF* (M_v ‘ *set xs*) ◁ M_v *x*)
by (*simp add*: *HF-insert-hinsert*)
also have ... = M_v *x* ⊔ ⨆ *HF* (M_v ‘ *set xs*) **by** *auto*
finally have I_{sa} $\mathcal{M}$ *a*
using *a* **by** *simp*
}
with *2.IH* **show** *?case* **by** *blast*
qed

lemma *soundness-introduce-v*:
∀ *fml* ∈ *introduce-v*. *interp* I_{sa} $\mathcal{M}$ *fml*
proof −
{fix α **assume** α ∈ P^+ *V*
have $\mathcal{M}$ v_α = ⨅ *HF* (M_v ‘ α) − ⨆ *HF* (M_v ‘ (*V* − α))
by *simp*
also have ... = ⨅ *HF* (($\mathcal{M}$ ∘ *Solo*) ‘ α) − ⨆ *HF* (($\mathcal{M}$ ∘ *Solo*) ‘ (*V* − α))
by *simp*
finally have I_{sa} $\mathcal{M}$ (*restriction-on-v* α)
apply (*simp add*: *set-V-list*)
using ‹α ∈ P^+ *V*›
by (*metis Int-def inf.absorb2 mem-P-plus-subset set-diff-eq*)
}
then have ∀ α ∈ P^+ *V*. *interp* I_{sa} $\mathcal{M}$ (*AT* (*restriction-on-v* α))
by *simp*
moreover
from *soundness-restriction-on-InterOfVars*
have ∀ *a* ∈ (*restriction-on-InterOfVars* ∘ *var-set-to-list*) α. I_{sa} $\mathcal{M}$ *a* **if** α ∈ P^+ *V* **for** α
by (*metis comp-apply mem-P-plus-subset set-var-set-to-list that*)
then have ∀ *lt* ∈ *AT* ‘ ⋃ ((*restriction-on-InterOfVars* ∘ *var-set-to-list*) ‘ P^+ *V*). *interp* I_{sa} $\mathcal{M}$ *lt*
by *fastforce*
moreover
from *soundness-restriction-on-UnionOfVars*
have ∀ *a* ∈ (*restriction-on-UnionOfVars* ∘ *var-set-to-list*) α. I_{sa} $\mathcal{M}$ *a* **if** α ∈ *Pow V* **for** α

by (*metis Pow-iff comp-apply set-var-set-to-list that*)
then have $\forall\, lt \in AT\ `\bigcup\ ((restriction\text{-}on\text{-}UnionOfVars \circ var\text{-}set\text{-}to\text{-}list)\ `\ Pow\ V).\ interp\ I_{sa}\ \mathcal{M}\ lt$
by *fastforce*
ultimately
show *?thesis*
unfolding *introduce-v-def* **by** *blast*
qed

lemma *soundness-restriction-on-UnionOfVennRegions*:
assumes *set* $\alpha s \in Pow\ (Pow\ V)$
shows $\forall\, a \in restriction\text{-}on\text{-}UnionOfVennRegions\ \alpha s.\ I_{sa}\ \mathcal{M}\ a$
proof (*induction* αs *rule*: *restriction-on-UnionOfVennRegions.induct*)
case *1*
then show *?case* **by** *auto*
next
case (*2* $\alpha\ \alpha s$)
{fix a **assume** $a \in restriction\text{-}on\text{-}UnionOfVennRegions\ (\alpha\ \#\ \alpha s) - restriction\text{-}on\text{-}UnionOfVennRegions\ \alpha s$
then have a: $a = Var\ (UnionOfVennRegions\ (\alpha\ \#\ \alpha s)) =_s Var\ v_\alpha \sqcup_s Var\ (UnionOfVennRegions\ \alpha s)$
by *fastforce*
have $\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ set\ (\alpha\ \#\ \alpha s)) = \bigsqcup HF\ (insert\ (\mathcal{M}\ v_\alpha)\ ((\mathcal{M} \circ VennRegion)\ `\ set\ \alpha s))$
by *simp*
also have $... = \bigsqcup\ (HF\ ((\mathcal{M} \circ VennRegion)\ `\ set\ \alpha s) \triangleleft \mathcal{M}\ v_\alpha)$
by (*simp add*: *HF-insert-hinsert*)
also have $... = \mathcal{M}\ v_\alpha \sqcup \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ set\ \alpha s)$
by *blast*
finally have $I_{sa}\ \mathcal{M}\ a$ **using** a **by** *simp*
}
with *2.IH* **show** *?case* **by** *blast*
qed

lemma *soundness-introduce-UnionOfVennRegions*:
$\forall\, lt \in introduce\text{-}UnionOfVennRegions.\ interp\ I_{sa}\ \mathcal{M}\ lt$
proof
fix *lt* **assume** $lt \in introduce\text{-}UnionOfVennRegions$
then obtain αs **where** $\alpha s \in set\ all\text{-}V\text{-}set\text{-}lists\ lt \in AT\ `\ restriction\text{-}on\text{-}UnionOfVennRegions\ \alpha s$
unfolding *introduce-UnionOfVennRegions-def* **by** *blast*
with *soundness-restriction-on-UnionOfVennRegions*
show $interp\ I_{sa}\ \mathcal{M}\ lt$
using *set-all-V-set-lists* **by** *fastforce*
qed

lemma *soundness-restriction-on-FunOfUnionOfVennRegions*:
assumes l'-l: $l' = var\text{-}set\text{-}set\text{-}to\text{-}var\text{-}set\text{-}list\ l$
and m'-m: $m' = var\text{-}set\text{-}set\text{-}to\text{-}var\text{-}set\text{-}list\ m$

shows $\exists\, lt \in set\ (restriction\text{-}on\text{-}FunOfUnionOfVennRegions\ l'\ m'\ f).\ interp\ I_{sa}\ \mathcal{M}\ lt$
proof $(cases\ \mathcal{M}\ (UnionOfVennRegions\ l') = \mathcal{M}\ (UnionOfVennRegions\ m'))$
case *True*
then have $\mathcal{M}\ w_{fl} = \mathcal{M}\ w_{fm}$
using *l'-l m'-m* **by** *auto*
then have $interp\ I_{sa}\ \mathcal{M}\ (AT\ (Var\ w_{fset\ l'} =_s\ Var\ w_{fset\ m'}))$
using *l'-l m'-m* **by** *auto*
then show *?thesis* **by** *simp*
next
case *False*
then have $interp\ I_{sa}\ \mathcal{M}\ (AF\ (Var\ (UnionOfVennRegions\ l') =_s\ Var\ (UnionOfVennRegions\ m')))$
by *fastforce*
then show *?thesis* **by** *simp*
qed

lemma *soundness-introduce-w*:
$\exists\, clause \in introduce\text{-}w.\ \forall\, lt \in clause.\ interp\ I_{sa}\ \mathcal{M}\ lt$
proof −
let $?f = \lambda lts.\ if\ interp\ I_{sa}\ \mathcal{M}\ (lts\ !\ 0)\ then\ lts\ !\ 0\ else\ lts\ !\ 1$
let $?g = \lambda(l,\ m,\ f).\ restriction\text{-}on\text{-}FunOfUnionOfVennRegions\ l\ m\ f$
let $?xs = List.product\ all\text{-}V\text{-}set\text{-}lists\ (List.product\ all\text{-}V\text{-}set\text{-}lists\ F\text{-}list)$
have $\forall\,(l',\ m',\ f) \in set\ ?xs.\ ?f\ (?g\ (l',\ m',\ f)) \in set\ (?g\ (l',\ m',\ f))$
by *fastforce*
with *valid-choice*[**where** $?f = ?f$ **and** $?g = ?g$ **and** $?xs = ?xs$]
have $map\ ?f\ (map\ ?g\ ?xs) \in set\ (choices\text{-}from\text{-}lists\ (map\ ?g\ ?xs))$
by *fast*
then have $set\ (map\ ?f\ (map\ ?g\ ?xs)) \in introduce\text{-}w$
unfolding *introduce-w-def*
using *mem-set-map*[**where** $?x = map\ ?f\ (map\ ?g\ ?xs)$ **and** $?f = set$]
by *blast*
moreover
have $\{x \in set\ V\text{-}set\text{-}list.\ x \in set\ l'\} = set\ l'$ **if** $l' \in set\ all\text{-}V\text{-}set\text{-}lists$ **for** l'
using *that set-V-set-list set-all-V-set-lists* **by** *auto*
then have $interp\ I_{sa}\ \mathcal{M}\ (?f\ (restriction\text{-}on\text{-}FunOfUnionOfVennRegions\ l'\ m'\ f))$
if $l' \in set\ all\text{-}V\text{-}set\text{-}lists\ m' \in set\ all\text{-}V\text{-}set\text{-}lists$ **for** $l'\ m'\ f$
using *that* **by** *auto*
then have $\forall\, lt \in set\ (map\ ?f\ (map\ ?g\ ?xs)).\ interp\ I_{sa}\ \mathcal{M}\ lt$
by *force*
ultimately
show *?thesis* **by** *blast*
qed

lemma *soundness-reduce-literal*:
assumes $lt \in set\ \mathcal{C}$
shows $\forall\, fml \in reduce\text{-}literal\ lt.\ interp\ I_{sa}\ \mathcal{M}\ fml$
proof −

from *norm-$\mathcal{C}$* ‹*lt* ∈ *set* $\mathcal{C}$› **have** *norm-literal lt* **by** *auto*
then show *?thesis*
proof (*cases rule*: *norm-literal.cases*)
case (*inc f*)
show *?thesis*
proof
fix *fml* **assume** *fml* ∈ *reduce-literal lt*
then have *fml* ∈ *reduce-literal* (AT_m (*inc*(*f*)))
using *inc* **by** *blast*
then obtain *l m* **where** *lm*: $l \subseteq P^+\ V\ m \subseteq P^+\ V\ l \subseteq m$
and *fml*: $fml = AT\ (Var\ w_{fm} =_s Var\ w_{fm} \sqcup_s Var\ w_{fl})$
by *auto*
from *model-for-$\mathcal{C}$* ‹*lt* ∈ *set* $\mathcal{C}$› *inc* **have** $I_a\ M_v\ M_f$ (*inc*(*f*)) **by** *fastforce*
then have $\forall s\ t.\ s \leq t \longrightarrow (M_f\ f)\ s \leq (M_f\ f)\ t$ **by** *simp*
moreover
from *lm* **have** $\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l) \leq \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ m)$
by (*metis HUnion-proper-Venn-region-inter* $\mathcal{M}$.*simps*(*2*) *comp-apply image-cong inf.absorb-iff2*)
ultimately
have $M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l)) \leq M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ m))$
by *blast*
then have $M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ m)) =$
$M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ m)) \sqcup M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l))$
by *blast*
with *fml lm* **show** *interp* $I_{sa}\ \mathcal{M}$ *fml*
by (*auto simp only*: *interp.simps* I_{sa}.*simps* I_{st}.*simps* $\mathcal{M}$.*simps set-var-set-set-to-var-set-list*)
qed
next
case (*dec f*)
show *?thesis*
proof
fix *fml* **assume** *fml* ∈ *reduce-literal lt*
then have *fml* ∈ *reduce-literal* (AT_m (*dec*(*f*)))
using *dec* **by** *blast*
then obtain *l m* **where** *lm*: $l \subseteq P^+\ V\ m \subseteq P^+\ V\ l \subseteq m$
and *fml*: $fml = AT\ (Var\ w_{fl} =_s Var\ w_{fl} \sqcup_s Var\ w_{fm})$
by *auto*
from *model-for-$\mathcal{C}$* ‹*lt* ∈ *set* $\mathcal{C}$› *dec* **have** $I_a\ M_v\ M_f$ (*dec*(*f*)) **by** *fastforce*
then have $\forall s\ t.\ s \leq t \longrightarrow (M_f\ f)\ t \leq (M_f\ f)\ s$ **by** *simp*
moreover
from *lm* **have** $\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l) \leq \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ m)$
by (*metis HUnion-proper-Venn-region-inter* $\mathcal{M}$.*simps*(*2*) *comp-apply image-cong inf.absorb-iff2*)
ultimately
have $M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ m)) \leq M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ$

$VennRegion)$ ‘ $l))$
by *blast*
then have $M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$ ‘ $l)) =$
$M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$ ‘ $l))\ \sqcup\ M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ$
$VennRegion)$ ‘ $m))$
by *blast*
with *fml lm* **show** *interp* $I_{sa}\ \mathcal{M}$ *fml*
by (*auto simp only*: *interp.simps* I_{sa}.*simps* I_{st}.*simps* $\mathcal{M}$.*simps set-var-set-set-to-var-set-list*)
qed
next
case (*add f*)
show *?thesis*
proof
fix *fml* **assume** *fml* ∈ *reduce-literal lt*
then have *fml* ∈ *reduce-literal* $(AT_m\ (add(f)))$
using *add* **by** *blast*
then obtain *l m* **where** *lm*: $l \subseteq P^+\ V\ m \subseteq P^+\ V$
and *fml*: $fml = AT\ (Var\ w_{fl \cup m} =_s\ Var\ w_{fl} \sqcup_s\ Var\ w_{fm})$
by *auto*
from *model-for-*$\mathcal{C}$ ‹*lt* ∈ *set* $\mathcal{C}$› *add* **have** $I_a\ M_v\ M_f\ (add(f))$ **by** *fastforce*
then have $\forall\, s\ t.\ (M_f\ f)\ (s \sqcup t) = (M_f\ f)\ s \sqcup (M_f\ f)\ t$ **by** *simp*
moreover
have $\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$ ‘ $(l \cup m)) = \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$ ‘ $l)$
$\sqcup \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$ ‘ $m)$
using *HUnion-proper-Venn-region-union* $\mathcal{M}$.*simps*(*2*) *lm*(*1*) *lm*(*2*) **by** *auto*
ultimately
have $M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$ ‘ $(l \cup m))) =$
$M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$ ‘ $l)) \sqcup M_f\ f\ (\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$
‘ $m))$
by *auto*
with *fml lm* **show** *interp* $I_{sa}\ \mathcal{M}$ *fml*
using *set-var-set-set-to-var-set-list*
apply (*simp only*: *interp.simps* I_{sa}.*simps* I_{st}.*simps* $\mathcal{M}$.*simps*)
by (*metis le-sup-iff*)
qed
next
case (*mul f*)
with *model-for-*$\mathcal{C}$ ‹*lt* ∈ *set* $\mathcal{C}$› **have** $I_a\ M_v\ M_f\ (mul(f))$ **by** *fastforce*
then have *f-mul*: $\forall\, s\ t.\ (M_f\ f)\ (s \sqcap t) = (M_f\ f)\ s \sqcap (M_f\ f)\ t$ **by** *simp*
have *InterOfWAux*: $I_{sa}\ \mathcal{M}\ (Var\ (InterOfWAux\ f\ l\ m) =_s\ Var\ w_{fl} -_s\ Var\ w_{fm})$
for *l m*
by *auto*
{fix *l m* **assume** $l \subseteq P^+\ V\ m \subseteq P^+\ V$
then have $\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$ ‘ $(l \cap m)) = \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$
‘ $l) \sqcap \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)$ ‘ $m)$
using *HUnion-proper-Venn-region-inter* **by** *force*
then have $\mathcal{M}\ (UnionOfVennRegions\ (var\text{-}set\text{-}set\text{-}to\text{-}var\text{-}set\text{-}list\ (l \cap m))) =$
$\mathcal{M}\ (UnionOfVennRegions\ (var\text{-}set\text{-}set\text{-}to\text{-}var\text{-}set\text{-}list\ l)) \sqcap$
$\mathcal{M}\ (UnionOfVennRegions\ (var\text{-}set\text{-}set\text{-}to\text{-}var\text{-}set\text{-}list\ m))$

```
      using set-var-set-set-to-var-set-list ‹l ⊆ P⁺ V› ‹m ⊆ P⁺ V›
      by (metis 𝓜.simps(3) inf.absorb-iff2 inf-left-commute)
    with f-mul have 𝓜 w_{fl∩m} = 𝓜 w_fl ⊓ 𝓜 w_fm
      by auto
    moreover
    from InterOfWAux have 𝓜 (InterOfWAux f l m) = 𝓜 w_fl − 𝓜 w_fm
      by simp
    ultimately
    have 𝓜 w_{fl∩m} = 𝓜 w_fl − 𝓜 (InterOfWAux f l m)
      by auto
    then have I_sa 𝓜 (Var w_{fl∩m} =_s Var w_fl −_s Var (InterOfWAux f l m))
      by auto
  }
  with InterOfWAux show ?thesis
    using mul by auto
next
  case (le f g)
  show ?thesis
  proof
    fix fml assume fml ∈ reduce-literal lt
    then have fml ∈ reduce-literal (AT_m (f ⪯_m g))
      using le by blast
    then obtain l where l: l ⊆ P⁺ V
                    and fml: fml = AT (Var w_gl =_s Var w_gl ⊔_s Var w_fl)
      by auto
    from model-for-𝒞 ‹lt ∈ set 𝒞› le have I_a M_v M_f (f ⪯_m g) by fastforce
    then have ∀ s. (M_f f) s ≤ (M_f g) s by simp
     then have M_f f (⨆ HF ((𝓜 ∘ VennRegion) ‘ l)) ≤ M_f g (⨆ HF ((𝓜 ∘ VennRegion) ‘ l))
      by auto
    with fml l show interp I_sa 𝓜 fml
      using set-var-set-set-to-var-set-list
      by (auto simp only: interp.simps I_sa.simps I_st.simps 𝓜.simps)
  qed
next
  case (eq-empty x n)
  with ‹lt ∈ set 𝒞› model-for-𝒞 have M_v x = 0 by auto
  show ?thesis
  proof
    fix fml assume fml ∈ reduce-literal lt
    with eq-empty have fml = AT (Var (Solo x) =_s ∅ n)
      by simp
    with ‹M_v x = 0› show interp I_sa 𝓜 fml by auto
  qed
next
  case (eq x y)
  with ‹lt ∈ set 𝒞› model-for-𝒞 have M_v x = M_v y by auto
  show ?thesis
  proof
```

```
    fix fml assume fml ∈ reduce-literal lt
    with eq have fml = AT (Var (Solo x) =s Var (Solo y))
      by simp
    with ‹Mv x = Mv y› show interp Isa M fml by auto
   qed
  next
   case (neq x y)
   with ‹lt ∈ set C› model-for-C have Mv x ≠ Mv y by auto
   show ?thesis
   proof
    fix fml assume fml ∈ reduce-literal lt
    with neq have fml = AF (Var (Solo x) =s Var (Solo y))
      by simp
    with ‹Mv x ≠ Mv y› show interp Isa M fml by auto
   qed
  next
   case (union x y z)
   with ‹lt ∈ set C› model-for-C have Mv x = Mv y ⊔ Mv z by fastforce
  then have interp Isa M (AT (Var (Solo x) =s Var (Solo y) ⊔s Var (Solo z)))
by simp
   with union show ?thesis by auto
  next
   case (diff x y z)
   with ‹lt ∈ set C› model-for-C have Mv x = Mv y − Mv z by fastforce
  then have interp Isa M (AT (Var (Solo x) =s Var (Solo y) −s Var (Solo z)))
by simp
   with diff show ?thesis by auto
  next
   case (single x y)
   with ‹lt ∈ set C› model-for-C have Mv x = HF {Mv y} by fastforce
    then have interp Isa M (AT (Var (Solo x) =s Single (Var (Solo y)))) by
simp
   with single show ?thesis by auto
  next
   case (app x f y)
   with ‹lt ∈ set C› model-for-C
   have Mv x = (Mf f) (Mv y) by fastforce
   moreover
   from app ‹lt ∈ set C› have y ∈ V
    unfolding V-def by force
   with variable-as-composition-of-proper-Venn-regions
   have Mv y = ⨆ HF (proper-Venn-region ‘ L V y)
    by presburger
   then have Mv y = ⨆ HF ((M ∘ VennRegion) ‘ L V y)
    by simp
   ultimately
   have M (Solo x) = M wfL V y
    using M.simps set-var-set-set-to-var-set-list L-subset-P-plus
    by metis
```

with *app* **show** *?thesis* **by** *simp*
qed
qed

lemma *soundness-reduce-cl*:
$\forall$ *fml* $\in$ *reduce-clause*. *interp* I_{sa} $\mathcal{M}$ *fml*
unfolding *reduce-clause-def*
using *soundness-reduce-literal*
by *fastforce*

lemma $\mathcal{M}$*-is-model-for-reduced-dnf*: *is-model-for-reduced-dnf* $\mathcal{M}$
unfolding *is-model-for-reduced-dnf-def*
unfolding *reduced-dnf-def*
using *soundness-introduce-v soundness-introduce-w soundness-introduce-UnionOfVennRegions soundness-reduce-cl*
by (*metis* (*no-types*, *lifting*) *Un-iff imageI*)

end

lemma *MLSSmf-to-MLSS-soundness*:
assumes $\mathcal{C}$*-norm*: *norm-clause* $\mathcal{C}$
and $\mathcal{C}$*-has-model*: $\exists\, M_v\ M_f.\ I_{cl}\ M_v\ M_f\ \mathcal{C}$
shows $\exists\, M.$ *normalized-MLSSmf-clause.is-model-for-reduced-dnf* $\mathcal{C}$ *M*
proof −
from $\mathcal{C}$*-has-model* **obtain** $M_v\ M_f$ **where** $I_{cl}\ M_v\ M_f\ \mathcal{C}$ **by** *blast*
with $\mathcal{C}$*-norm*
interpret *satisfiable-normalized-MLSSmf-clause* $\mathcal{C}$ M_v M_f
by *unfold-locales*
from $\mathcal{M}$*-is-model-for-reduced-dnf* **show** *?thesis* **by** *auto*
qed

end
theory *Reduced-MLSS-Formula-Singleton-Model-Property*
imports *Syntactic-Description Place-Realisation MLSSmf-to-MLSS*
begin

locale *satisfiable-normalized-MLSS-clause-with-vars-for-proper-Venn-regions* =
satisfiable-normalized-MLSS-clause $\mathcal{C}$ $\mathcal{A}$ **for** $\mathcal{C}$ $\mathcal{A}$ +
fixes *U* :: *'a set*
— The collection of variables representing the proper Venn regions of the "original" variable set of the MLSSmf clause
assumes *U-subset-V*: $U \subseteq V$
and *no-overlap-within-U*: $[\![u_1 \in U;\ u_2 \in U;\ u_1 \neq u_2]\!] \Longrightarrow \mathcal{A}\ u_1 \sqcap \mathcal{A}\ u_2 = 0$
and *U-collect-places-neq*: $AF\ (Var\ x =_s Var\ y) \in \mathcal{C} \Longrightarrow$
$\exists\, L\ M.\ L \subseteq U \wedge M \subseteq U \wedge \mathcal{A}\ x = \bigsqcup HF\ (\mathcal{A}\ `\ L) \wedge \mathcal{A}\ y = \bigsqcup HF\ (\mathcal{A}\ `\ M)$
and *U-collect-places-single*: $AT\ (Var\ x =_s Single\ (Var\ y)) \in \mathcal{C} \Longrightarrow$
$\exists\, L\ M.\ L \subseteq U \wedge M \subseteq U \wedge \mathcal{A}\ x = \bigsqcup HF\ (\mathcal{A}\ `\ L) \wedge \mathcal{A}\ y = \bigsqcup HF\ (\mathcal{A}\ `\ M)$
begin

interpretation $\mathfrak{B}$: *adequate-place-framework* $\mathcal{C}$ *PI* at_p
using *syntactic-description-is-adequate* **by** *blast*

lemma *fact-1*:
assumes $u_1 \in U$
and $u_2 \in U$
and $u_1 \neq u_2$
and $\pi \in PI$
shows $\neg\ (\pi\ u_1 \wedge \pi\ u_2)$
proof (*rule ccontr*)
assume $\neg \neg\ (\pi\ u_1 \wedge \pi\ u_2)$
then have $\pi\ u_1\ \pi\ u_2$ **by** *blast+*
from ‹$\pi \in PI$› **obtain** σ **where** $\sigma \in \Sigma\ \pi = \pi_\sigma$ **by** *auto*
then have $\sigma \neq 0$ **by** *fastforce*
from ‹$\pi = \pi_\sigma$› ‹$\pi\ u_1$› ‹$\pi\ u_2$› **have** $\sigma \leq \mathcal{A}\ u_1\ \sigma \leq \mathcal{A}\ u_2$ **by** *simp+*
with ‹$\sigma \neq 0$› **have** $\mathcal{A}\ u_1 \sqcap \mathcal{A}\ u_2 \neq 0$ **by** *blast*
with *no-overlap-within-U* **show** *False*
using ‹$u_1 \in U$› ‹$u_2 \in U$› ‹$u_1 \neq u_2$› **by** *blast*
qed

fun *place-eq* :: ($'a \Rightarrow bool$) $\Rightarrow$ ($'a \Rightarrow bool$) $\Rightarrow$ *bool* **where**
place-eq $\pi_1\ \pi_2 \longleftrightarrow (\forall\, x \in V.\ \pi_1\ x = \pi_2\ x)$

fun *place-sim* :: ($'a \Rightarrow bool$) $\Rightarrow$ ($'a \Rightarrow bool$) $\Rightarrow$ *bool* (**infixl** $\sim$ *50*) **where**
place-sim $\pi_1\ \pi_2 \longleftrightarrow$ *place-eq* $\pi_1\ \pi_2 \vee (\exists\, u \in U.\ \pi_1\ u \wedge \pi_2\ u)$

abbreviation *rel-place-sim* $\equiv \{(\pi_1, \pi_2) \in PI \times PI.\ \pi_1 \sim \pi_2\}$

lemma *place-sim-rel-equiv-on-PI*: *equiv PI rel-place-sim*
proof (*rule equivI*)
have *rel-place-sim* $\subseteq PI \times PI$ **by** *blast*
moreover
have $(\pi, \pi) \in$ *rel-place-sim* **if** $\pi \in PI$ **for** π
using *that* **by** *fastforce*
ultimately
show *refl-on PI rel-place-sim* **using** *refl-onI* **by** *blast*

show *sym rel-place-sim*
proof (*rule symI*)
fix $\pi_1\ \pi_2$ **assume** $(\pi_1, \pi_2) \in$ *rel-place-sim*
then have $\pi_1 \in PI\ \pi_2 \in PI\ \pi_1 \sim \pi_2$ **by** *blast+*
then show $(\pi_2, \pi_1) \in$ *rel-place-sim* **by** *auto*
qed

show *trans rel-place-sim*
proof (*rule transI*)
fix $\pi_1\ \pi_2\ \pi_3$
assume $(\pi_1, \pi_2) \in$ *rel-place-sim* $(\pi_2, \pi_3) \in$ *rel-place-sim*
then have $\pi_1 \in PI\ \pi_2 \in PI\ \pi_3 \in PI\ \pi_1 \sim \pi_2\ \pi_2 \sim \pi_3$ **by** *blast+*

then consider *place-eq* $\pi_1\ \pi_2 \wedge$ *place-eq* $\pi_2\ \pi_3$ | *place-eq* $\pi_1\ \pi_2 \wedge (\exists\, u \in U.\ \pi_2\ u \wedge \pi_3\ u)$
| $(\exists\, u \in U.\ \pi_1\ u \wedge \pi_2\ u) \wedge$ *place-eq* $\pi_2\ \pi_3$ | $(\exists\, u \in U.\ \pi_1\ u \wedge \pi_2\ u) \wedge (\exists\, u \in U.\ \pi_2\ u \wedge \pi_3\ u)$
by *auto*
then have $\pi_1 \sim \pi_3$
proof (*cases*)
case *1*
then have *place-eq* $\pi_1\ \pi_3$ **by** *auto*
then show *?thesis* **by** *auto*
next
case *2*
then obtain u **where** $u \in U\ \pi_2\ u\ \pi_3\ u$ **by** *blast*
with *U-subset-V* **have** $u \in V$ **by** *blast*
with *2* **have** $\pi_1\ u \longleftrightarrow \pi_2\ u$ **by** *force*
with ‹$\pi_2\ u$› **have** $\pi_1\ u$ **by** *blast*
with ‹$u \in U$› ‹$\pi_3\ u$›
show *?thesis* **by** *auto*
next
case *3*
then obtain u **where** $u \in U\ \pi_1\ u\ \pi_2\ u$ **by** *blast*
with *U-subset-V* **have** $u \in V$ **by** *blast*
with *3* **have** $\pi_2\ u \longleftrightarrow \pi_3\ u$ **by** *force*
with ‹$\pi_2\ u$› **have** $\pi_3\ u$ **by** *blast*
with ‹$u \in U$› ‹$\pi_1\ u$›
show *?thesis* **by** *auto*
next
case *4*
then obtain $u_1\ u_2$ **where** $u_1 \in U\ \pi_1\ u_1\ \pi_2\ u_1$ **and** $u_2 \in U\ \pi_2\ u_2\ \pi_3\ u_2$
by *blast*
with *fact-1* **have** $u_1 = u_2$
using ‹$\pi_2 \in PI$› **by** *blast*
with ‹$\pi_3\ u_2$› **have** $\pi_3\ u_1$ **by** *blast*
with ‹$\pi_1\ u_1$› ‹$u_1 \in U$› **show** *?thesis*
by *auto*
qed
with ‹$\pi_1 \in PI$› ‹$\pi_2 \in PI$› ‹$\pi_3 \in PI$›
show $(\pi_1, \pi_3) \in$ *rel-place-sim* **by** *blast*
qed
qed *auto*

lemma *refl-sim*:
assumes $a \in PI$
and $b \in PI$
and $a \sim b$
shows $b \sim a$
using *assms* **by** *auto*

lemma *trans-sim*:

```
  assumes a ∈ PI
     and b ∈ PI
     and c ∈ PI
     and a ∼ b
     and b ∼ c
    shows a ∼ c
proof −
  from assms have (a, b) ∈ rel-place-sim (b, c) ∈ rel-place-sim
    by blast+
  with place-sim-rel-equiv-on-PI have (a, c) ∈ rel-place-sim
    using equivE transE
    by (smt (verit, ccfv-SIG))
  then show a ∼ c by blast
qed

lemma fact-2:
  assumes x ∈ V
     and exL: ∃ L ⊆ U. 𝒜 x = ⨆ HF (𝒜 ‘ L)
     and π1 ∈ PI
     and π2 ∈ PI
     and π1 ∼ π2
    shows π1 x ⟷ π2 x
proof (cases place-eq π1 π2)
  case True
  with ‹x ∈ V› show ?thesis by force
next
  case False
  with ‹π1 ∼ π2› obtain u where u ∈ U π1 u π2 u by auto
  from exL obtain L where L ⊆ U 𝒜 x = ⨆ HF (𝒜 ‘ L) by blast
  from ‹L ⊆ U› U-subset-V finite-V have finite L
    by (simp add: finite-subset)

  have π x ⟷ u ∈ L if π u π ∈ PI for π
  proof −
    from ‹π ∈ PI› obtain σ where π = πσ σ ∈ Σ by auto
    with ‹π u› have σ ≤ 𝒜 u
      using ‹u ∈ U› U-subset-V by auto
    have σ ≤ 𝒜 x ⟷ u ∈ L
    proof (standard)
      assume σ ≤ 𝒜 x
      {assume u ∉ L
       then have ∀ v ∈ L. v ≠ u by blast
       with no-overlap-within-U have ∀ v ∈ L. 𝒜 v ⊓ 𝒜 u = 0
         using ‹L ⊆ U› ‹u ∈ U› by auto
       with ‹σ ≤ 𝒜 u› have ∀ v ∈ L. 𝒜 v ⊓ σ = 0 by blast
       then have ⨆ HF (𝒜 ‘ L) ⊓ σ = 0
         using finite-V U-subset-V ‹L ⊆ U› by auto
       with ‹𝒜 x = ⨆ HF (𝒜 ‘ L)› have 𝒜 x ⊓ σ = 0 by argo
       with ‹σ ≤ 𝒜 x› have False
```

```
        using ‹σ ∈ Σ› mem-Σ-not-empty by blast
      }
      then show u ∈ L by blast
    next
      assume u ∈ L
      with ‹σ ≤ 𝒜 u› have σ ≤ ⨆ HF (𝒜 ‘ L)
        using ‹finite L› by force
      with ‹𝒜 x = ⨆ HF (𝒜 ‘ L)› show σ ≤ 𝒜 x by simp
    qed
    with ‹π = π_σ› show π x ⟷ u ∈ L
      using ‹x ∈ V› associated-place.simps by blast
  qed
  with ‹π₁ ∈ PI› ‹π₁ u› ‹π₂ ∈ PI› ‹π₂ u›
  have π₁ x ⟷ u ∈ L π₂ x ⟷ u ∈ L by blast+
  then show ?thesis by blast
qed

lemma U-collect-places-single': y ∈ W ⟹ ∃ L. L ⊆ U ∧ 𝒜 y = ⨆ HF (𝒜 ‘ L)
  using U-collect-places-single
  by (meson memW-E)

definition PI' :: ('a ⇒ bool) set where
  PI' ≡ (λπs. SOME π. π ∈ πs) ‘ (PI // rel-place-sim)

definition rep :: ('a ⇒ bool) ⇒ ('a ⇒ bool) where
  rep π = (SOME π'. π' ∈ rel-place-sim ‘‘ {π})

lemma range-rep:
  assumes π ∈ PI
    shows rep π ∈ PI'
  using assms
  unfolding PI'-def rep-def
  using quotientI[where ?x = π and ?A = PI and ?r = rel-place-sim]
  by blast

lemma PI'-eq-image-of-rep-on-PI: PI' = rep ‘ PI
proof (standard; standard)
  fix π assume π ∈ PI'
  then obtain πs where πs ∈ PI // rel-place-sim π = (SOME π. π ∈ πs)
    unfolding PI'-def by blast
  then obtain π₀ where πs = rel-place-sim ‘‘ {π₀} π₀ ∈ PI
    using quotientE[where ?A = PI and ?r = rel-place-sim and ?X = πs]
    by blast
  with ‹π = (SOME π. π ∈ πs)› have π = rep π₀
    unfolding rep-def by blast
 with ‹π₀ ∈ PI› show π ∈ rep ‘ PI by blast
next
  fix π assume π ∈ rep ‘ PI
  then obtain π₀ where π₀ ∈ PI π = rep π₀ by blast
```

then show $\pi \in PI'$ **using** *range-rep* **by** *blast*
qed

lemma *rep-sim*:
assumes $\pi \in PI$
shows $\pi \sim rep\ \pi$
and $rep\ \pi \sim \pi$
proof −
from ‹$\pi \in PI$› **have** $\pi \in$ *rel-place-sim* `` $\{\pi\}$ **by** *fastforce*
then obtain π' **where** $\pi' = rep\ \pi$ **by** *blast*
with *someI*[*of* $\lambda x.\ x \in$ *rel-place-sim* `` $\{\pi\}$] **have** $\pi' \in$ *rel-place-sim* `` $\{\pi\}$
using ‹$\pi \in$ *rel-place-sim* `` $\{\pi\}$›
unfolding *rep-def* **by** *fast*
with ‹$\pi' = rep\ \pi$› **show** $\pi \sim rep\ \pi$ **by** *fast*
with *place-sim-rel-equiv-on-PI* **show** $rep\ \pi \sim \pi$
by (*metis* (*full-types*) *place-eq.simps place-sim.elims*(*1*))
qed

lemma *PI'-subset-PI*: $PI' \subseteq PI$
unfolding *PI'-def*
using *equiv-Eps-preserves place-sim-rel-equiv-on-PI* **by** *blast*

lemma *sim-self*:
assumes $\pi \in PI'$
and $\pi' \in PI'$
and $\pi \sim \pi'$
shows $\pi' = \pi$
proof −
from ‹$\pi \sim \pi'$› **have** $(\pi, \pi') \in$ *rel-place-sim*
using ‹$\pi \in PI'$› ‹$\pi' \in PI'$› *PI'-subset-PI* **by** *blast*
from ‹$\pi \in PI'$› **obtain** πs **where** $\pi s \in PI\ //$ *rel-place-sim* $\pi = (SOME\ \pi.\ \pi \in \pi s)$
unfolding *PI'-def* **by** *blast*
then have $\pi \in \pi s$
using *equiv-Eps-in place-sim-rel-equiv-on-PI* **by** *blast*
from ‹$\pi' \in PI'$› **obtain** $\pi s'$ **where** $\pi s' \in PI\ //$ *rel-place-sim* $\pi' = (SOME\ \pi.\ \pi \in \pi s')$
unfolding *PI'-def* **by** *blast*
then have $\pi' \in \pi s'$
using *equiv-Eps-in place-sim-rel-equiv-on-PI* **by** *blast*
from *place-sim-rel-equiv-on-PI* ‹$\pi s \in PI\ //$ *rel-place-sim*› ‹$\pi s' \in PI\ //$ *rel-place-sim*›
‹$\pi \in \pi s$› ‹$\pi' \in \pi s'$› ‹$(\pi, \pi') \in$ *rel-place-sim*›
have $\pi s = \pi s'$
using *quotient-eqI*[**where** *?A* $= PI$ **and** *?r* $=$ *rel-place-sim* **and** *?x* $= \pi$ **and** *?X* $= \pi s$ **and** *?y* $= \pi'$ **and** *?Y* $= \pi s'$]
by *fast*
with ‹$\pi = (SOME\ \pi.\ \pi \in \pi s)$› ‹$\pi' = (SOME\ \pi.\ \pi \in \pi s')$› **show** $\pi' = \pi$
by *auto*
qed

fun $at_p\text{-}f' :: 'a \Rightarrow ('a \Rightarrow bool)$ **where**
  $at_p\text{-}f'\ w = rep\ (at_p\text{-}f\ w)$

definition $at_p' = \{(y,\ at_p\text{-}f'\ y) | y.\ y \in W\}$
declare $at_p'\text{-}def$ [*simp*]

lemma *range-*at_p*-*f':
  assumes $w \in W$
  shows $at_p\text{-}f'\ w \in PI'$
proof −
  from ‹$w \in W$› *range-*at_p*-*f **have** $at_p\text{-}f\ w \in PI$ **by** *blast*
  then have *rel-place-sim* `` $\{at_p\text{-}f\ w\} \in PI$ // *rel-place-sim*
    using *quotientI* **by** *fast*
  then show *?thesis* **unfolding** $PI'\text{-}def$
    apply (*simp only*: $at_p\text{-}f'.simps$ *rep-def*)
   by (*smt* (*verit*, *best*) *Eps-cong* $at_p\text{-}f'.elims$ *image-insert insert-iff mk-disjoint-insert*)
qed

lemma *rep-at*:
  assumes $\pi \in PI$
      and $(y,\ \pi) \in at_p$
    shows $(y,\ rep\ \pi) \in at_p'$
proof −
  from ‹$(y,\ \pi) \in at_p$› **have** $at_p\text{-}f\ y = \pi$ **by** *auto*
  from ‹$(y,\ \pi) \in at_p$› **have** $y \in W$ **by** *auto*
  with *W-subset-V* **have** $y \in V$ **by** *fast*
  from ‹$(y,\ \pi) \in at_p$› **obtain** x **where** $AT\ (Var\ x =_s Single\ (Var\ y)) \in \mathcal{C}\ x \in V$
    using *memW-E* **by** *fastforce*
  with *U-collect-places-single* **have** $\exists L.\ L \subseteq U \wedge \mathcal{A}\ x = \bigsqcup HF\ (\mathcal{A}\ `\ L)$ **by** *meson*
  with *fact-2* **have** $\pi_1\ x \longleftrightarrow \pi_2\ x$ **if** $\pi_1 \sim \pi_2\ \pi_1 \in PI\ \pi_2 \in PI$ **for** $\pi_1\ \pi_2$
    using ‹$x \in V$› *that* **by** *blast*
  with *rep-sim* **have** $(rep\ \pi)\ x \longleftrightarrow \pi\ x$
    using $PI'\text{-}subset\text{-}PI$ ‹$\pi \in PI$› *range-rep* **by** *blast*

  from $\mathfrak{B}.C5\text{-}1$[**where** *?x* = x **and** *?y* = y] **have** $\pi\ x\ \forall \pi' \in PI.\ \pi' \neq \pi \longrightarrow \neg\ \pi'\ x$
    using ‹$AT\ (Var\ x =_s Single\ (Var\ y)) \in \mathcal{C}$› ‹$(y,\ \pi) \in at_p$› **by** *fastforce*+

  from ‹$\pi\ x$› ‹$(rep\ \pi)\ x \longleftrightarrow \pi\ x$› **have** $(rep\ \pi)\ x$ **by** *blast*
  with ‹$\forall \pi' \in PI.\ \pi' \neq \pi \longrightarrow \neg\ \pi'\ x$› **have** $rep\ \pi = \pi$
    using *range-rep* $PI'\text{-}subset\text{-}PI$ ‹$\pi \in PI$› **by** *blast*
  then have $at_p\text{-}f'\ y = rep\ \pi$
    using ‹$at_p\text{-}f\ y = \pi$› **by** (*simp only*: $at_p\text{-}f'.simps$)
  then show $(y,\ rep\ \pi) \in at_p'$
    using ‹$y \in W$›
    by (*metis* (*mono-tags*, *lifting*) $at_p'\text{-}def$ *mem-Collect-eq*)
qed

interpretation $\mathfrak{B}'$: *adequate-place-framework* $\mathcal{C}\ PI'\ at_p'$

proof −
from *PI′-subset-PI* $\mathfrak{B}$.*PI-subset-places-V*
have *PI′-subset-places-V*: $PI' \subseteq places\ V$ **by** *blast*

have *dom-at*$_p$′: $Domain\ at_p' = W$ **by** *auto*
have *range-at*$_p$′: $Range\ at_p' \subseteq PI'$
proof −
{**fix** *y lt* **assume** $lt \in \mathcal{C}$ $y \in singleton\text{-}vars\ lt$
then have $rep\ (at_p\text{-}f\ y) \in PI'$
using *range-at*$_p$*-f*[*of y*] *range-rep*[*of at*$_p$*-f y*]
by *blast*
}
then show *?thesis* **by** *auto*
qed

from $\mathfrak{B}$.*single-valued-at*$_p$
have *single-valued-at*$_p$′: $single\text{-}valued\ at_p'$
unfolding *single-valued-def at*$_p$′*-def*
apply (*simp only*: *at*$_p$*-f′.simps*)
by *blast*

from *PI′-subset-PI* **have** $place\text{-}membership\ \mathcal{C}\ PI' \subseteq place\text{-}membership\ \mathcal{C}\ PI$ **by** *auto*
with $\mathfrak{B}$.*membership-irreflexive* **have** *membership-irreflexive*:
$(\pi, \pi) \notin place\text{-}membership\ \mathcal{C}\ PI'$ **for** π
by *blast*

from *PI′-subset-PI* **have** *subgraph*: $subgraph\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI')\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI)$
proof −
have $verts\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI') = PI'$ **by** *simp*
moreover
have $verts\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI) = PI$ **by** *simp*
ultimately
have *verts*: $verts\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI') \subseteq verts\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI)$
using *PI′-subset-PI* **by** *presburger*

have $arcs\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI') = place\text{-}membership\ \mathcal{C}\ PI'$ **by** *simp*
moreover
have $arcs\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI) = place\text{-}membership\ \mathcal{C}\ PI$ **by** *simp*
moreover
have $place\text{-}membership\ \mathcal{C}\ PI' \subseteq place\text{-}membership\ \mathcal{C}\ PI$
using *PI′-subset-PI* **by** *auto*
ultimately
have *arcs*: $arcs\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI') \subseteq arcs\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI)$ **by** *blast*

have $compatible\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI)\ (place\text{-}mem\text{-}graph\ \mathcal{C}\ PI')$
unfolding *compatible-def* **by** *simp*

with *verts arcs* **show** *subgraph* (*place-mem-graph* $\mathcal{C}$ *PI'*) (*place-mem-graph* $\mathcal{C}$ *PI*)
unfolding *subgraph-def*
using *place-mem-graph-wf-digraph*
by *blast*
qed

from $\mathfrak{B}$.*C6* **have** $\nexists c.$ *pre-digraph.cycle* (*place-mem-graph* $\mathcal{C}$ *PI*) *c*
using *dag.acyclic* **by** *blast*
then have $\nexists c.$ *pre-digraph.cycle* (*place-mem-graph* $\mathcal{C}$ *PI'*) *c*
using *subgraph wf-digraph.subgraph-cycle* **by** *blast*
then have *C6*: *dag* (*place-mem-graph* $\mathcal{C}$ *PI'*)
using ‹*dag* (*place-mem-graph* $\mathcal{C}$ *PI*)› *dag-axioms-def dag-def digraph.digraph-subgraph subgraph*
by *blast*

from $\mathfrak{B}$.*C1-1 PI'-subset-PI*
have *C1-1*: $\exists n.$ *AT* (*Var x* $=_s$ $\emptyset$ *n*) $\in \mathcal{C} \Longrightarrow \forall \pi \in PI'. \neg \pi\ x$ **for** *x*
by *fast*

from $\mathfrak{B}$.*C1-2 PI'-subset-PI*
have *C1-2*: *AT* (*Var x* $=_s$ *Var y*) $\in \mathcal{C} \Longrightarrow \forall \pi \in PI'. \pi\ x \longleftrightarrow \pi\ y$ **for** *x y*
by *fast*

from $\mathfrak{B}$.*C2 PI'-subset-PI*
have *C2*: *AT* (*Var x* $=_s$ *Var y* $\sqcup_s$ *Var z*) $\in \mathcal{C} \Longrightarrow \forall \pi \in PI'. \pi\ x \longleftrightarrow \pi\ y \vee \pi\ z$ **for** *x y z*
by *fast*

from $\mathfrak{B}$.*C3 PI'-subset-PI*
have *C3*: *AT* (*Var x* $=_s$ *Var y* $-_s$ *Var z*) $\in \mathcal{C} \Longrightarrow \forall \pi \in PI'. \pi\ x \longleftrightarrow \pi\ y \wedge \neg \pi\ z$ **for** *x y z*
by *fast*

have *C4*: *AF* (*Var x* $=_s$ *Var y*) $\in \mathcal{C} \Longrightarrow \exists \pi \in PI'. \pi\ x \longleftrightarrow \neg \pi\ y$ **for** *x y*
proof −
assume *neq*: *AF* (*Var x* $=_s$ *Var y*) $\in \mathcal{C}$
with $\mathfrak{B}$.*C4* **obtain** π **where** $\pi \in PI\ \pi\ x \longleftrightarrow \neg \pi\ y$ **by** *blast*
from *neq* **have** $x \in V\ y \in V$ **by** *fastforce+*
from *neq U-collect-places-neq*[**where** *?x* = *x* **and** *?y* = *y*] *fact-2*[*of x*]
have *sim-π-x*: $\pi_1\ x = \pi_2\ x$ **if** $\pi_1 \in PI\ \pi_2 \in PI\ \pi_1 \sim \pi_2$ **for** $\pi_1\ \pi_2$
using *that* ‹$x \in V$› **by** *blast*
from *neq U-collect-places-neq*[**where** *?x* = *x* **and** *?y* = *y*] *fact-2*[*of y*]
have *sim-π-y*: $\pi_1\ y = \pi_2\ y$ **if** $\pi_1 \in PI\ \pi_2 \in PI\ \pi_1 \sim \pi_2$ **for** $\pi_1\ \pi_2$
using *that* ‹$y \in V$› **by** *blast*
from ‹$\pi \in PI$› **have** *rep* $\pi \in PI'$ **using** *range-rep* **by** *blast*
then have *rep* $\pi \in PI$ **using** *PI'-subset-PI* **by** *blast*

from *rep-sim sim-π-x* **have** (*rep* π) $x \longleftrightarrow \pi\ x$

```
    using ‹rep π ∈ PI› ‹π ∈ PI› by blast
  moreover
  from rep-sim sim-π-y have π y ⟷ (rep π) y
    using ‹rep π ∈ PI› ‹π ∈ PI› by blast
  ultimately
  have (rep π) x ⟷ ¬ (rep π) y
    using ‹π x ⟷ ¬ π y› by blast
  with ‹rep π ∈ PI'› show ?thesis by blast
 qed

 have C5-1: ∃π. (y, π) ∈ at_p' ∧ π x ∧ (∀π' ∈ PI'. π' ≠ π ⟶ ¬ π' x)
  if AT (Var x =_s Single (Var y)) ∈ 𝒞 for x y
 proof −
  from that have y ∈ W x ∈ V y ∈ V by fastforce+
  from that 𝔅.C5-1[where ?y = y and ?x = x]
  obtain π where π: (y, π) ∈ at_p π x ∀π' ∈ PI. π' ≠ π ⟶ ¬ π' x
    by blast
  with 𝔅.range-at_p have π ∈ PI by fast
  then have rep π ∈ PI' using range-rep by blast
  from rep-sim have rep π ∼ π using ‹π ∈ PI› by fast
  with U-collect-places-single ‹π x› fact-2 have (rep π) x
    using ‹x ∈ V› ‹π ∈ PI› ‹rep π ∈ PI'› PI'-subset-PI that
    by blast
  with π have rep π = π
    using ‹rep π ∈ PI'› PI'-subset-PI by blast
  with π show ?thesis
    using ‹rep π ∈ PI'› PI'-subset-PI
    by (metis rep-at subset-iff)
 qed

 have C5-2: ∀π ∈ PI'. π y ⟷ π z if y ∈ W z ∈ W and at'-eq: ∃π. (y, π) ∈
at_p' ∧ (z, π) ∈ at_p' for y z
 proof
  fix π assume π ∈ PI'
  from at'-eq obtain π' where π': at_p-f' y = π' at_p-f' z = π'
    by (simp only: at_p'-def) fast
  with range-at_p-f' ‹y ∈ W› have π' ∈ PI' by blast
  from π' have at_p-f' y ∼ at_p-f' z
    apply (simp only: at_p-f'.simps place-sim.simps place-eq.simps)
    by blast
  moreover
  from rep-sim have at_p-f' y ∼ at_p-f y
    using at_p-f'.simps range-at_p-f that(1) by presburger
  moreover
  from rep-sim have at_p-f' z ∼ at_p-f z
    using at_p-f'.simps range-at_p-f that(2) by presburger
  ultimately
  have at_p-f y ∼ at_p-f z
    using trans-sim[of at_p-f y at_p-f' y at_p-f' z]
```

using *trans-sim*[*of* at_p-f y at_p-f' z at_p-f z]
using *refl-sim*[*of* at_p-f' y at_p-f y]
using *range-*at_p*-*f[*of* y] *range-*at_p*-*f[*of* z] *range-*at_p*-*f' *PI'-subset-PI that*(*1*−*2*)
by (*meson subset-iff*)
then consider at_p-f y = at_p-f z | $\exists\, u \in U.$ at_p-f y u $\wedge$ at_p-f z u
by *force*
then show π $y \longleftrightarrow \pi$ z
proof (*cases*)
case *1*
then have $\exists\, \pi.$ $(y, \pi) \in at_p \wedge (z, \pi) \in at_p$
using at_p*-def* ‹$y \in W$› ‹$z \in W$› **by** *blast*
with $\mathfrak{B}$.*C5-2* **have** $\forall\, \pi \in PI.$ π $y \longleftrightarrow \pi$ z
using ‹$y \in W$› ‹$z \in W$› **by** *presburger*
with ‹$\pi \in PI'$› *PI'-subset-PI* **show** π $y \longleftrightarrow \pi$ z
by *fast*
next
case *2*
then obtain u **where** $u \in U$ at_p-f y u at_p-f z u **by** *blast*
then have $\mathcal{A}$ $y \in \mathcal{A}$ u $\mathcal{A}$ $z \in \mathcal{A}$ u
by (*simp add*: *less-eq-hf-def*)+
from ‹$y \in W$› **obtain** x_1 **where** x_1*-single-y*: AT (Var x_1 $=_s$ $Single$ (Var y)) $\in \mathcal{C}$
using *memW-E* **by** *blast*
with $\mathcal{A}$*-sat-*$\mathcal{C}$ **have** $\mathcal{A}$ x_1 = HF {$\mathcal{A}$ y} **by** *fastforce*
then have $\mathcal{A}$ $y \in \mathcal{A}$ x_1 **by** *simp*
from x_1*-single-y U-collect-places-single* **obtain** L **where** $L \subseteq U$ $\mathcal{A}$ $x_1 = \bigsqcup HF$ ($\mathcal{A}$ ‘ L)
by *meson*
with ‹$\mathcal{A}$ $y \in \mathcal{A}$ x_1› **obtain** u' **where** $u' \in L$ $\mathcal{A}$ $y \in \mathcal{A}$ u' **by** *auto*
from ‹$\mathcal{A}$ $x_1 = \bigsqcup HF$ ($\mathcal{A}$ ‘ L)› ‹$u' \in L$› **have** $\mathcal{A}$ $u' \leq \mathcal{A}$ x_1
using ‹$\mathcal{A}$ $y \in \mathcal{A}$ x_1› **by** *auto*
with ‹$\mathcal{A}$ x_1 = HF {$\mathcal{A}$ y}› ‹$\mathcal{A}$ $y \in \mathcal{A}$ u'› **have** $\mathcal{A}$ u' = HF {$\mathcal{A}$ y} **by** *auto*
with ‹$\mathcal{A}$ $y \in \mathcal{A}$ u› ‹$u \in U$› ‹$u' \in L$› ‹$L \subseteq U$› *no-overlap-within-U*
have u' = u **by** *fastforce*
with ‹$\mathcal{A}$ u' = HF {$\mathcal{A}$ y}› ‹$\mathcal{A}$ $z \in \mathcal{A}$ u› **have** $\mathcal{A}$ y = $\mathcal{A}$ z **by** *simp*
with *realise-same-implies-eq-under-all-*π[*of* y z π] **show** *?thesis*
using ‹$y \in W$› ‹$z \in W$› *W-subset-V* ‹$\pi \in PI'$› *PI'-subset-PI* **by** *blast*
qed
qed

have *C5-3*: $\exists\, \pi.$ $(y, \pi) \in at_p' \wedge (y', \pi) \in at_p'$
if $y \in W$ $y' \in W$ $\forall\, \pi' \in PI'.$ π' $y' \longleftrightarrow \pi'$ y **for** y' y
proof −
from ‹$\forall\, \pi' \in PI'.$ π' $y' \longleftrightarrow \pi'$ y› **have** $\forall\, \pi \in PI.$ *rep* π $y' \longleftrightarrow$ *rep* π y
by (*metis range-rep*)
{**fix** π **assume** $\pi \in PI$
with ‹$\forall\, \pi' \in PI'.$ π' $y' \longleftrightarrow \pi'$ y› **have** *rep* π $y' \longleftrightarrow$ *rep* π y
using *range-rep* **by** *fast*
from ‹$\pi \in PI$› *PI'-subset-PI range-rep* **have** *rep* $\pi \in PI$ **by** *blast*

from *U-collect-places-single'*[*of* y'] *fact-2*[*of* y' *rep* π π] *rep-sim*[*of* π]
have *rep* π $y' \longleftrightarrow \pi$ y'
using ‹$y' \in W$› *W-subset-V* ‹$\pi \in PI$› ‹*rep* $\pi \in PI$›
by *blast*
from *U-collect-places-single'*[*of* y] *fact-2*[*of* y *rep* π π] *rep-sim*[*of* π]
have *rep* π $y \longleftrightarrow \pi$ y
using ‹$y \in W$› *W-subset-V* ‹$\pi \in PI$› ‹*rep* $\pi \in PI$›
by *blast*
from ‹*rep* π $y' \longleftrightarrow$ *rep* π y› ‹*rep* π $y' \longleftrightarrow \pi$ y'› ‹*rep* π $y \longleftrightarrow \pi$ y›
have π $y \longleftrightarrow \pi$ y' **by** *blast*
}
with $\mathfrak{B}$.*C5-3* **obtain** π **where** $(y, \pi) \in at_p$ $(y', \pi) \in at_p$
using ‹$y \in W$› ‹$y' \in W$› **by** *blast*
then have $(y,$ *rep* $\pi) \in at_p{}'$ $(y',$ *rep* $\pi) \in at_p{}'$
by (*meson Range-iff* $\mathfrak{B}$.*range-at*$_p$ *rep-at subset-iff*)+
then show *?thesis* **by** *fast*
qed

have $\pi = \pi_{HF\ \{0\}}$ **if** $\pi \in$ *Range* $at_p{}' -$ *Range* (*place-membership* $\mathcal{C}$ PI') **for** π
proof −
from *that* **obtain** y **where** $(y, \pi) \in at_p{}'$ **by** *blast*
then have $y \in W$ $\pi \in PI'$
using *dom-at*$_p{}'$ *range-at*$_p{}'$ **by** *blast*+
from ‹$(y, \pi) \in at_p{}'$› **have** $\pi =$ *rep* (*at*$_p$-*f* y) **by** *simp*
from ‹$y \in W$› **obtain** x **where** *lt-in-C*: AT (*Var* $x =_s$ *Single* (*Var* y)) $\in \mathcal{C}$
using *memW-E* **by** *blast*
with *A-sat-C* **have** $\mathcal{A}$ $x = HF$ $\{\mathcal{A}$ $y\}$ **by** *fastforce*
then have $\sigma_y \leq \mathcal{A}$ x **by** *simp*
with *lt-in-C* **have** *at*$_p$-*f* y x **by** *force*
with ‹$\pi =$ *rep* (*at*$_p$-*f* y)› *fact-2*[*of* x] *rep-sim*[*of at*$_p$-*f* y] *U-collect-places-single*[*of* x y]
have π x
using *lt-in-C* ‹$\pi \in PI'$› *PI'-subset-PI* ‹$y \in W$›
by (*smt* (*verit*, *best*) $\mathfrak{B}$.*PI-subset-places-V places-domain range-at*$_p$*-f rev-contra-hsubsetD*)

have $\forall \pi \in PI.\ \neg\ \pi$ y
proof (*rule ccontr*)
assume $\neg$ ($\forall \pi \in PI.\ \neg\ \pi$ y)
then obtain π' **where** $\pi' \in PI$ π' y **by** *blast*
with *U-collect-places-single'*[*of* y] *fact-2*[*of* y *rep* π' π'] *rep-sim*[*of* π']
have *rep* π' y
using ‹$y \in W$› *PI'-subset-PI W-subset-V range-rep* **by** *blast*
with ‹AT (*Var* $x =_s$ *Single* (*Var* y)) $\in \mathcal{C}$› ‹π x›
have (*rep* π', π) $\in$ *place-membership* $\mathcal{C}$ PI'
using ‹$\pi \in PI'$› ‹$\pi' \in PI$› *range-rep*
by (*simp only*: *place-membership.simps*) *blast*
then have $\pi \in$ *Range* (*place-membership* $\mathcal{C}$ PI') **by** *blast*
with *that* **show** *False* **by** *blast*
qed

```
    have ∀ α ∈ ℒ V y. proper-Venn-region α = 0
    proof (rule ccontr)
      assume ¬ (∀ α ∈ ℒ V y. proper-Venn-region α = 0)
      then obtain α where α: α ∈ ℒ V y proper-Venn-region α ≠ 0 by blast
      then have y ∈ α α ∈ P⁺ V by auto
      with ‹proper-Venn-region α ≠ 0› have proper-Venn-region α ≤ 𝒜 y
        using proper-Venn-region-subset-variable-iff
        by (meson mem-P-plus-subset subset-iff)
      then have π_{proper-Venn-region α y}
        using W-subset-V ‹y ∈ W› by auto
      with ‹∀ π ∈ PI. ¬ π y› show False
        using α by auto
    qed
    then have ⨆ HF (proper-Venn-region ` ℒ V y) = 0
      by fastforce
    with variable-as-composition-of-proper-Venn-regions[of y]
    have 𝒜 y = 0
      using ‹y ∈ W› W-subset-V by auto
    with ‹𝒜 x = HF {𝒜 y}› have 𝒜 x = HF {0} by argo

    from ‹π ∈ PI'› PI'-subset-PI obtain σ where σ ∈ Σ π = π_σ
      by (metis PI-def image-iff in-mono)
    with ‹π x› have σ ≠ 0 σ ≤ 𝒜 x by simp+
    with ‹𝒜 x = HF {0}› have σ = HF {0} by fastforce
    with ‹π = π_σ› show π = π_{HF {0}} by blast
  qed
  then have C7:
    ⟦π₁ ∈ Range at_p' − Range (place-membership 𝒞 PI');
      π₂ ∈ Range at_p' − Range (place-membership 𝒞 PI')⟧ ⟹ π₁ = π₂ for π₁ π₂
    by blast

  from PI'-subset-places-V dom-at_p' range-at_p' single-valued-at_p'
       membership-irreflexive C6
       C1-1 C1-2 C2 C3 C4 C5-1 C5-2 C5-3 C7
  show adequate-place-framework 𝒞 PI' at_p'
    apply intro-locales
    unfolding adequate-place-framework-axioms-def
    by blast
qed

lemma singleton-model-for-normalized-reduced-literals:
  ∃ 𝓜. ∀ lt ∈ 𝒞. interp I_sa 𝓜 lt ∧ (∀ u ∈ U. hcard (𝓜 u) ≤ 1)
proof −
  from 𝔅'.finite-PI have finite (PI' − Range at_p') by blast
  with u-exists[of PI' − Range at_p' card PI'] obtain u where
    ⟦π₁ ∈ PI' − Range at_p'; π₂ ∈ PI' − Range at_p'; π₁ ≠ π₂⟧ ⟹ u π₁ ≠ u π₂
    π ∈ PI' − Range at_p' ⟹ hcard (u π) ≥ card PI'
  for π₁ π₂ π
    by blast
```

```
  then have place-realization 𝒞 PI' at_p' u
    by unfold-locales blast+

  {fix x assume x ∈ U
    then have π₁ = π₂ if π₁ x π₂ x π₁ ∈ PI' π₂ ∈ PI' for π₁ π₂
      using sim-self that by auto
    then consider {π ∈ PI'. π x} = {} | (∃π. {π ∈ PI'. π x} = {π})
      by blast
    then have hcard (place-realization.ℳ 𝒞 PI' at_p' u x) ≤ 1
    proof (cases)
      case 1
      then have place-realization.ℳ 𝒞 PI' at_p' u x = 0
        using ‹place-realization 𝒞 PI' at_p' u› place-realization.ℳ.simps
        by fastforce
      then show ?thesis by simp
    next
      case 2
      then obtain π where {π ∈ PI'. π x} = {π} π ∈ PI' by auto
    then have place-realization.ℳ 𝒞 PI' at_p' u x = ⨆ HF (place-realization.place-realise
𝒞 PI' at_p' u ‘ {π})
        using ‹place-realization 𝒞 PI' at_p' u› place-realization.ℳ.simps
        by fastforce
      also have ... = ⨆ HF {place-realization.place-realise 𝒞 PI' at_p' u π}
        by simp
    finally have place-realization.ℳ 𝒞 PI' at_p' u x = ⨆ HF {place-realization.place-realise
𝒞 PI' at_p' u π} .
      moreover
      from place-realization.place-realise-singleton[of 𝒞 PI' at_p' u]
      have hcard (place-realization.place-realise 𝒞 PI' at_p' u π) = 1
        using ‹place-realization 𝒞 PI' at_p' u› ‹π ∈ PI'› by blast
     then obtain c where place-realization.place-realise 𝒞 PI' at_p' u π = HF {c}
        using hcard-1E[of place-realization.place-realise 𝒞 PI' at_p' u π]
        by fastforce
      ultimately
      have place-realization.ℳ 𝒞 PI' at_p' u x = ⨆ HF {HF {c}}
        by presburger
      also have ... = HF {c} by fastforce
      also have hcard ... = 1
        by (simp add: hcard-def)
      finally show ?thesis by linarith
    qed
  }
  moreover
  from place-realization.ℳ-sat-𝒞
  have ∀ lt ∈ 𝒞. interp I_sa (place-realization.ℳ 𝒞 PI' at_p' u) lt
    using ‹place-realization 𝒞 PI' at_p' u› by fastforce
  ultimately
  show ?thesis by blast
qed
```

end

theorem *singleton-model-for-reduced-MLSS-clause*:
assumes *norm-*$\mathcal{C}$: *normalized-MLSSmf-clause* $\mathcal{C}$
and V: $V = vars_m\ \mathcal{C}$
and $\mathcal{A}$*-model*: *normalized-MLSSmf-clause.is-model-for-reduced-dnf* $\mathcal{C}\ \mathcal{A}$
shows $\exists \mathcal{M}.$ *normalized-MLSSmf-clause.is-model-for-reduced-dnf* $\mathcal{C}\ \mathcal{M} \wedge$
$(\forall \alpha \in P^+\ V.$ *hcard* $(\mathcal{M}\ v_\alpha) \leq 1)$
proof −
from *norm-*$\mathcal{C}$ **interpret** *normalized-MLSSmf-clause* $\mathcal{C}$ **by** *blast*
interpret *proper-Venn-regions* $V\ \mathcal{A} \circ$ *Solo*
using V **by** *unfold-locales blast*

from $\mathcal{A}$*-model* **have** $\forall$ *fm*∈*introduce-v*. *interp* $I_{sa}\ \mathcal{A}$ *fm*
unfolding *is-model-for-reduced-dnf-def reduced-dnf-def*
by *blast*
with *eval-v* **have** $\mathcal{A}$*-v*: $\forall \alpha \in P^+\ V.\ \mathcal{A}\ v_\alpha =$ *proper-Venn-region* α
using V *V-def proper-Venn-region.simps* **by** *auto*

from $\mathcal{A}$*-model* **have** $\forall$ *lt* ∈ *introduce-UnionOfVennRegions*. *interp* $I_{sa}\ \mathcal{A}$ *lt*
unfolding *is-model-for-reduced-dnf-def reduced-dnf-def* **by** *blast*
then have $\forall$ *a* ∈ *restriction-on-UnionOfVennRegions* αs. $I_{sa}\ \mathcal{A}\ a$
if $\alpha s \in$ *set all-V-set-lists* **for** αs
unfolding *introduce-UnionOfVennRegions-def*
using *that* **by** *simp*
with *eval-UnionOfVennRegions* **have** $\mathcal{A}$*-UnionOfVennRegions*:
$\mathcal{A}$ (*UnionOfVennRegions* αs) = $\bigsqcup$ *HF* ($\mathcal{A}$ ‘ *VennRegion* ‘ *set* αs)
if $\alpha s \in$ *set all-V-set-lists* **for** αs
using *that* **by** (*simp add*: *Sup.SUP-image*)

have *Solo-variable-as-composition-of-v*:
$\exists L \subseteq \{v_\alpha\ |\alpha.\ \alpha \in P^+\ V\}.\ \mathcal{A}\ z = \bigsqcup$ *HF* ($\mathcal{A}$ ‘ L) **if** $\exists z' \in V.\ z =$ *Solo* z' **for** z
proof −
from *that* **obtain** z' **where** $z' \in V\ z =$ *Solo* z' **by** *blast*
then have *VennRegion* ‘ $\mathcal{L}\ V\ z' \subseteq \{v_\alpha\ |\alpha.\ \alpha \in P^+\ V\}$ **by** *fastforce*
moreover
from $\mathcal{A}$*-v* **have** $\forall \alpha \in \mathcal{L}\ V\ z'.\ \mathcal{A}\ v_\alpha =$ *proper-Venn-region* α
using $\mathcal{L}$*-subset-P-plus finite-V* **by** *fast*
then have $\bigsqcup$ *HF* ($\mathcal{A}$ ‘ (*VennRegion* ‘ $\mathcal{L}\ V\ z'$)) = $\bigsqcup$ *HF* (*proper-Venn-region* ‘ $\mathcal{L}\ V\ z'$)
using *HUnion-eq*[**where** *?S* = $\mathcal{L}\ V\ z'$ **and** *?f* = $\mathcal{A} \circ$ *VennRegion* **and** *?g* = *proper-Venn-region*]
by (*simp add*: *image-comp*)
moreover
from *variable-as-composition-of-proper-Venn-regions*
have ($\mathcal{A} \circ$ *Solo*) z' = $\bigsqcup$ *HF* (*proper-Venn-region* ‘ $\mathcal{L}\ V\ z'$)
using ‹$z' \in V$› **by** *presburger*
with ‹$z =$ *Solo* z'› **have** $\mathcal{A}\ z = \bigsqcup$ *HF* (*proper-Venn-region* ‘ $\mathcal{L}\ V\ z'$) **by** *simp*

ultimately
have *VennRegion* ‘ $\mathcal{L}$ V z' $\subseteq$ $\{v_\alpha$ $|\alpha.$ $\alpha \in P^+$ $V\}$ $\wedge$ $\mathcal{A}$ $z = \bigsqcup$ *HF* ($\mathcal{A}$ ‘ *VennRegion* ‘ $\mathcal{L}$ V z')
by *simp*
then show *?thesis* **by** *blast*
qed

from $\mathcal{A}$*-model* **obtain** *clause* **where** *clause*:
clause $\in$ *reduced-dnf* $\forall$ *lt* $\in$ *clause*. *interp* I_{sa} $\mathcal{A}$ *lt*
unfolding *is-model-for-reduced-dnf-def* **by** *blast*
with *reduced-dnf-normalized* **have** *normalized-MLSS-clause clause* **by** *blast*
with *clause*
have *satisfiable-normalized-MLSS-clause-with-vars-for-proper-Venn-regions clause* $\mathcal{A}$ $\{v_\alpha$ $|\alpha.$ $\alpha \in P^+$ $V\}$
proof (*unfold-locales*, *goal-cases*)
case *1*
then show *?case*
using *normalized-MLSS-clause.norm-*$\mathcal{C}$ **by** *blast*
next
case *2*
then show *?case*
by (*simp add*: *normalized-MLSS-clause.finite-*$\mathcal{C}$)
next
case *3*
then show *?case*
by (*simp add*: *finite-vars-fm normalized-MLSS-clause.finite-*$\mathcal{C}$)
next
case *4*
then show *?case* **by** *simp*
next
case *5*
from ‹*clause* $\in$ *reduced-dnf*› *normalized-clause-contains-all-v-*α
have $\forall \alpha \in P^+$ $V.$ $v_\alpha \in \bigcup$ (*vars* ‘ *clause*)
using V *V-def* **by** *simp*
then show *?case* **by** *blast*
next
case (*6 x y*)
then obtain α β **where** $\alpha\beta$: $\alpha \in P^+$ V $\beta \in P^+$ V $x = v_\alpha$ $y = v_\beta$
by *blast*
with ‹$x \neq y$› **have** $\alpha \neq \beta$ **by** *blast*

from $\alpha\beta$ **have** $\alpha \subseteq V$ $\beta \subseteq V$ **by** *auto*

from $\mathcal{A}$*-model* **have** $\forall$ *fm*$\in$*introduce-v*. *interp* I_{sa} $\mathcal{A}$ *fm*
unfolding *is-model-for-reduced-dnf-def reduced-dnf-def* **by** *blast*
with $\alpha\beta$ *eval-v* **have** $\mathcal{A}$ x = *proper-Venn-region* α $\mathcal{A}$ y = *proper-Venn-region* β
using V *V-def proper-Venn-region.simps* **by** *auto*
with *proper-Venn-region-disjoint* ‹$\alpha \neq \beta$›
show *?case*

```
    using ‹α ⊆ V› ‹β ⊆ V› by presburger
  next
    case (7 x y)
    from ‹AF (Var x =s Var y) ∈ clause› ‹clause ∈ reduced-dnf›
    consider AF (Var x =s Var y) ∈ reduce-clause | ∃ clause ∈ introduce-w. AF
(Var x =s Var y) ∈ clause
      unfolding reduced-dnf-def introduce-v-def introduce-UnionOfVennRegions-def
by blast
    then show ?case
    proof (cases)
      case 1
      then obtain lt where lt: lt ∈ set C AF (Var x =s Var y) ∈ reduce-literal lt
        unfolding reduce-clause-def by blast
      then obtain a where lt = AFm a
        by (cases lt rule: reduce-literal.cases) auto
      from ‹lt ∈ set C› norm-C have norm-literal lt by blast
      with ‹lt = AFm a› norm-literal-neq
      obtain x' y' where lt: lt = AFm (Varm x' =m Varm y') by blast
      then have reduce-literal lt = {AF (Var (Solo x') =s Var (Solo y'))}
        by simp
      with ‹AF (Var x =s Var y) ∈ reduce-literal lt› have x = Solo x' y = Solo y'
        by simp+
      from lt ‹lt ∈ set C› have x' ∈ V y' ∈ V
        using V by fastforce+

      from Solo-variable-as-composition-of-v show ?thesis
        using ‹x = Solo x'› ‹y = Solo y'› ‹x' ∈ V› ‹y' ∈ V›
        by (smt (verit, best))
    next
      case 2
      with lt-in-clause-in-introduce-w-E obtain l' m' f
        where l': l' ∈ set all-V-set-lists
          and m': m' ∈ set all-V-set-lists
          and f: f ∈ set F-list
        and AF (Var x =s Var y) ∈ set (restriction-on-FunOfUnionOfVennRegions
l' m' f)
        by blast
      then have AF (Var x =s Var y) = AF (Var (UnionOfVennRegions l') =s
Var (UnionOfVennRegions m'))
        by auto
      then have x = UnionOfVennRegions l' y = UnionOfVennRegions m' by
blast+
      with A-UnionOfVennRegions l' m'
      have A x = ⋃ HF (A ‘ VennRegion ‘ set l') A y = ⋃ HF (A ‘ VennRegion ‘
set m')
        by blast+
      moreover
      from l' set-all-V-set-lists have set l' ⊆ P+ V
        using V V-def by auto
```

```
      then have VennRegion ‘ set l' ⊆ {v_α |α. α ∈ P⁺ V}
        by blast
      moreover
      from m' set-all-V-set-lists have set m' ⊆ P⁺ V
        using V V-def by auto
      then have VennRegion ‘ set m' ⊆ {v_α |α. α ∈ P⁺ V}
        by blast
      ultimately
      show ?thesis by blast
    qed
  next
    case (8 x y)
    then consider AT (Var x =_s Single (Var y)) ∈ introduce-v
      | ∃ clause ∈ introduce-w. AT (Var x =_s Single (Var y)) ∈ clause
      | AT (Var x =_s Single (Var y)) ∈ introduce-UnionOfVennRegions
      | AT (Var x =_s Single (Var y)) ∈ reduce-clause
      unfolding reduced-dnf-def by blast
    then show ?case
    proof (cases)
      case 1
      have Var x =_s Single (Var y) ≠ restriction-on-v α for α
        by simp
      moreover
      have Var x =_s Single (Var y) ∉ restriction-on-InterOfVars xs for xs
        by (induction xs rule: restriction-on-InterOfVars.induct) auto
    then have Var x =_s Single (Var y) ∉ (restriction-on-InterOfVars ∘ var-set-to-list)
α for α
        by simp
      moreover
      have Var x =_s Single (Var y) ∉ restriction-on-UnionOfVars xs for xs
        by (induction xs rule: restriction-on-UnionOfVars.induct) auto
         then have Var x =_s Single (Var y) ∉ (restriction-on-UnionOfVars ∘
var-set-to-list) α for α
        by simp
      ultimately
      have AT (Var x =_s Single (Var y)) ∉ introduce-v
        unfolding introduce-v-def by blast
      with 1 show ?thesis by blast
    next
      case 2
      with lt-in-clause-in-introduce-w-E obtain l' m' f
     where AT (Var x =_s Single (Var y)) ∈ set (restriction-on-FunOfUnionOfVennRegions
l' m' f)
        by blast
      moreover
    have AT (Var x =_s Single (Var y)) ∉ set (restriction-on-FunOfUnionOfVennRegions
l' m' f)
        by simp
      ultimately
```

```
    show ?thesis by blast
  next
    case 3
    have Var x =s Single (Var y) ∉ restriction-on-UnionOfVennRegions αs for
αs
      by (induction αs rule: restriction-on-UnionOfVennRegions.induct) auto
    then have AT (Var x =s Single (Var y)) ∉ introduce-UnionOfVennRegions
      unfolding introduce-UnionOfVennRegions-def by blast
    with 3 show ?thesis by blast
  next
    case 4
    then obtain lt where lt ∈ set C and reduce-lt: AT (Var x =s Single (Var
y)) ∈ reduce-literal lt
      unfolding reduce-clause-def by blast
    with norm-C have norm-literal lt by blast
    then have ∃ x' y'. lt = AT_m (Var_m x' =_m Single_m (Var_m y'))
      apply (cases lt rule: norm-literal.cases)
      using reduce-lt by auto
    then obtain x' y' where lt: lt = AT_m (Var_m x' =_m Single_m (Var_m y')) by
blast
    with reduce-lt have x = Solo x' y = Solo y' by simp+
    from ‹lt ∈ set C› lt have x' ∈ V y' ∈ V
      using V by fastforce+
    from Solo-variable-as-composition-of-v show ?thesis
      using ‹x = Solo x'› ‹y = Solo y'› ‹x' ∈ V› ‹y' ∈ V›
      by (smt (verit, best))
  qed
 qed
 then show ?thesis
  using satisfiable-normalized-MLSS-clause-with-vars-for-proper-Venn-regions.singleton-model-for-normalized
   unfolding is-model-for-reduced-dnf-def
  by (smt (verit) V V-def clause(1) introduce-v-subset-reduced-fms mem-Collect-eq
subset-iff v-α-in-vars-introduce-v)
qed

end
theory MLSSmf-to-MLSS-Completeness
 imports MLSSmf-Semantics MLSSmf-to-MLSS MLSSmf-HF-Extras
        Proper-Venn-Regions Reduced-MLSS-Formula-Singleton-Model-Property
begin

locale MLSSmf-to-MLSS-complete =
 normalized-MLSSmf-clause C for C :: ('v, 'f) MLSSmf-clause +
   fixes B :: ('v, 'f) Composite ⇒ hf
 assumes B: is-model-for-reduced-dnf B

   fixes Λ :: hf ⇒ 'v set set
 assumes Λ-subset-V: Λ x ⊆ P+ V
     and Λ-preserves-zero: Λ 0 = {}
```

```
    and Λ-inc: a ≤ b ⟹ Λ a ⊆ Λ b
    and Λ-add: Λ (a ⊔ b) = Λ a ∪ Λ b
    and Λ-mul: Λ (a ⊓ b) = Λ a ∩ Λ b
    and Λ-discr: l ⊆ P⁺ V ⟹
          a = ⨆ HF ((B ∘ VennRegion) ` l) ⟹ a = ⨆ HF ((B ∘ VennRegion)
` (Λ a))
begin

fun discretize_v :: (('v, 'f) Composite ⇒ hf) ⇒ ('v ⇒ hf) where
  discretize_v M = M ∘ Solo

fun discretize_f :: (('v, 'f) Composite ⇒ hf) ⇒ ('f ⇒ hf ⇒ hf) where
  discretize_f M = (λf a. M w_fΛ a)

interpretation proper-Venn-regions V discretize_v B
  using finite-V by unfold-locales

lemma all-literal-sat: ∀ lt ∈ set C. I_l (discretize_v B) (discretize_f B) lt
proof
  fix lt assume lt ∈ set C

  from B obtain clause where clause: clause ∈ reduced-dnf
                      and B-sat-clause: ∀ lt ∈ clause. interp I_sa B lt
    unfolding is-model-for-reduced-dnf-def by blast

  from ‹lt ∈ set C› have norm-literal lt
    using norm-C by blast
  then show I_l (discretize_v B) (discretize_f B) lt
  proof (cases lt rule: norm-literal.cases)
    case (inc f)
    have s ≤ t ⟹ discretize_f B f s ≤ discretize_f B f t for s t
    proof −
      let ?atom = Var w_fΛ t =_s Var w_fΛ t ⊔_s Var w_fΛ s
      assume s ≤ t
      then have Λ s ⊆ Λ t using Λ-inc by simp
      then have ?atom ∈ reduce-atom (inc(f))
        using Λ-subset-V
        by (simp add: V-def)
      then have AT ?atom ∈ clause
        using ‹lt = AT_m (inc(f))› ‹lt ∈ set C› clause
        unfolding reduced-dnf-def reduce-clause-def by fastforce
      with B-sat-clause have I_sa B ?atom by fastforce
      then have B w_fΛ t = B w_fΛ t ⊔ B w_fΛ s by simp
      then have B w_fΛ s ≤ B w_fΛ t
        by (simp add: sup.order-iff)
      then show discretize_f B f s ≤ discretize_f B f t by simp
    qed
    then show ?thesis using inc by auto
```

next
case (*dec f*)
have $s \leq t \Longrightarrow$ *discretize*$_f$ $\mathcal{B}$ *f t* $\leq$ *discretize*$_f$ $\mathcal{B}$ *f s* **for** *s t*
proof −
let *?atom* = *Var* $w_{f\Lambda\ s}$ $=_s$ *Var* $w_{f\Lambda\ s}$ $\sqcup_s$ *Var* $w_{f\Lambda\ t}$
assume $s \leq t$
then have $\Lambda\ s \subseteq \Lambda\ t$ **using** Λ-*inc* **by** *simp*
then have *?atom* $\in$ *reduce-atom* (*dec*(*f*))
using Λ-*subset-V*
by (*simp add*: *V-def*)
then have *AT ?atom* $\in$ *clause*
using ‹*lt* = *AT*$_m$ (*dec*(*f*))› ‹*lt* $\in$ *set* $\mathcal{C}$› *clause*
unfolding *reduced-dnf-def reduce-clause-def* **by** *fastforce*
with $\mathcal{B}$-*sat-clause* **have** I_{sa} $\mathcal{B}$ *?atom* **by** *fastforce*
then have $\mathcal{B}$ $w_{f\Lambda\ s} = \mathcal{B}$ $w_{f\Lambda\ s} \sqcup \mathcal{B}$ $w_{f\Lambda\ t}$ **by** *simp*
then have $\mathcal{B}$ $w_{f\Lambda\ t} \leq \mathcal{B}$ $w_{f\Lambda\ s}$
by (*simp add*: *sup.order-iff*)
then show *discretize*$_f$ $\mathcal{B}$ *f t* $\leq$ *discretize*$_f$ $\mathcal{B}$ *f s* **by** *simp*
qed
then show *?thesis* **using** *dec* **by** *auto*

next
case (*add f*)
have *discretize*$_f$ $\mathcal{B}$ *f* (*s* $\sqcup$ *t*) = *discretize*$_f$ $\mathcal{B}$ *f s* $\sqcup$ *discretize*$_f$ $\mathcal{B}$ *f t* **for** *s t*
proof −
let *?atom* = *Var* $w_{f\Lambda\ (s \sqcup t)}$ $=_s$ *Var* $w_{f\Lambda\ s}$ $\sqcup_s$ *Var* $w_{f\Lambda\ t}$
have *?atom* $\in$ *reduce-atom* (*add*(*f*))
using Λ-*subset-V* Λ-*add*
by (*simp add*: *V-def*)
then have *AT ?atom* $\in$ *clause*
using ‹*lt* = *AT*$_m$ (*add*(*f*))› ‹*lt* $\in$ *set* $\mathcal{C}$› *clause*
unfolding *reduced-dnf-def reduce-clause-def* **by** *fastforce*
with $\mathcal{B}$-*sat-clause* **have** I_{sa} $\mathcal{B}$ *?atom* **by** *fastforce*
then have $\mathcal{B}$ $w_{f\Lambda\ (s \sqcup t)} = \mathcal{B}$ $w_{f\Lambda\ s} \sqcup \mathcal{B}$ $w_{f\Lambda\ t}$ **by** *simp*
then show *discretize*$_f$ $\mathcal{B}$ *f* (*s* $\sqcup$ *t*) = *discretize*$_f$ $\mathcal{B}$ *f s* $\sqcup$ *discretize*$_f$ $\mathcal{B}$ *f t* **by** *simp*
qed
then show *?thesis* **using** *add* **by** *auto*

next
case (*mul f*)
have *discretize*$_f$ $\mathcal{B}$ *f* (*s* $\sqcap$ *t*) = *discretize*$_f$ $\mathcal{B}$ *f s* $\sqcap$ *discretize*$_f$ $\mathcal{B}$ *f t* **for** *s t*
proof −
let *?atom-1* = *Var* (*InterOfWAux f* (Λ *s*) (Λ *t*)) $=_s$ *Var* $w_{f\Lambda\ s}$ $-_s$ *Var* $w_{f\Lambda\ t}$
have *?atom-1* $\in$ *reduce-atom* (*mul*(*f*))
using Λ-*subset-V*
by (*simp add*: *V-def*)
then have *AT ?atom-1* $\in$ *clause*
using ‹*lt* = *AT*$_m$ (*mul*(*f*))› ‹*lt* $\in$ *set* $\mathcal{C}$› *clause*

```
      unfolding reduced-dnf-def reduce-clause-def by fastforce
    with B-sat-clause have I_sa B ?atom-1 by fastforce
    then have B (InterOfWAux f (Λ s) (Λ t)) = B w_fΛ s − B w_fΛ t by simp
    moreover
    let ?atom-2 = Var w_fΛ (s ⊓ t) =_s Var w_fΛ s −_s Var (InterOfWAux f (Λ s)
(Λ t))
    have ?atom-2 ∈ reduce-atom (mul(f))
      using Λ-subset-V Λ-mul
      by (simp add: V-def)
    then have AT ?atom-2 ∈ clause
      using ‹lt = AT_m (mul(f))› ‹lt ∈ set C› clause
      unfolding reduced-dnf-def reduce-clause-def by fastforce
    with B-sat-clause have I_sa B ?atom-2 by fastforce
     then have B w_fΛ (s ⊓ t) = B w_fΛ s − B (InterOfWAux f (Λ s) (Λ t)) by
simp
    ultimately
    have B w_fΛ (s ⊓ t) = B w_fΛ s ⊓ B w_fΛ t by auto
    then show discretize_f B f (s ⊓ t) = discretize_f B f s ⊓ discretize_f B f t by
simp
  qed
  then show ?thesis using mul by auto

next
  case (le f g)
  have discretize_f B f s ≤ discretize_f B g s for s
  proof −
    let ?atom = Var w_gΛ s =_s Var w_gΛ s ⊔_s Var w_fΛ s
    have ?atom ∈ reduce-atom (f ⪯_m g)
      using Λ-subset-V
      by (simp add: V-def)
    then have AT ?atom ∈ clause
      using ‹lt = AT_m (f ⪯_m g)› ‹lt ∈ set C› clause
      unfolding reduced-dnf-def reduce-clause-def by fastforce
    with B-sat-clause have I_sa B ?atom by fastforce
    then have B w_gΛ s = B w_gΛ s ⊔ B w_fΛ s by simp
    then have B w_fΛ s ≤ B w_gΛ s
      by (simp add: sup.orderI)
    then show discretize_f B f s ≤ discretize_f B g s by simp
  qed
  then show ?thesis using le by auto

next
  case (eq-empty x n)
  let ?lt = AT (Var (Solo x) =_s ∅ n)
  from eq-empty have ?lt ∈ reduce-literal lt
    using ‹lt ∈ set C› by simp
  then have ?lt ∈ clause
    using ‹lt ∈ set C› clause
    unfolding reduced-dnf-def reduce-clause-def by fastforce
```

with $\mathcal{B}$*-sat-clause* **have** *interp* I_{sa} $\mathcal{B}$ *?lt* **by** *fastforce*
with *eq-empty* **show** *?thesis* **by** *simp*

next
case (*eq x y*)
let *?lt* = *AT* (*Var* (*Solo x*) =$_s$ *Var* (*Solo y*))
from *eq* **have** *?lt* ∈ *reduce-literal lt*
using ‹*lt* ∈ *set* $\mathcal{C}$› **by** *simp*
then have *?lt* ∈ *clause*
using ‹*lt* ∈ *set* $\mathcal{C}$› *clause*
unfolding *reduced-dnf-def reduce-clause-def* **by** *fastforce*
with $\mathcal{B}$*-sat-clause* **have** *interp* I_{sa} $\mathcal{B}$ *?lt* **by** *fastforce*
with *eq* **show** *?thesis* **by** *simp*

next
case (*neq x y*)
let *?lt* = *AF* (*Var* (*Solo x*) =$_s$ *Var* (*Solo y*))
from *neq* **have** *?lt* ∈ *reduce-literal lt*
using ‹*lt* ∈ *set* $\mathcal{C}$› **by** *simp*
then have *?lt* ∈ *clause*
using ‹*lt* ∈ *set* $\mathcal{C}$› *clause*
unfolding *reduced-dnf-def reduce-clause-def* **by** *fastforce*
with $\mathcal{B}$*-sat-clause* **have** *interp* I_{sa} $\mathcal{B}$ *?lt* **by** *fastforce*
with *neq* **show** *?thesis* **by** *simp*

next
case (*union x y z*)
let *?lt* = *AT* (*Var* (*Solo x*) =$_s$ *Var* (*Solo y*) ⊔$_s$ *Var* (*Solo z*))
from *union* **have** *?lt* ∈ *reduce-literal lt*
using ‹*lt* ∈ *set* $\mathcal{C}$› **by** *simp*
then have *?lt* ∈ *clause*
using *neq* ‹*lt* ∈ *set* $\mathcal{C}$› *clause*
unfolding *reduced-dnf-def reduce-clause-def* **by** *fastforce*
with $\mathcal{B}$*-sat-clause* **have** *interp* I_{sa} $\mathcal{B}$ *?lt* **by** *fastforce*
with *union* **show** *?thesis* **by** *simp*

next
case (*diff x y z*)
let *?lt* = *AT* (*Var* (*Solo x*) =$_s$ *Var* (*Solo y*) −$_s$ *Var* (*Solo z*))
from *diff* **have** *?lt* ∈ *reduce-literal lt*
using ‹*lt* ∈ *set* $\mathcal{C}$› **by** *simp*
then have *?lt* ∈ *clause*
using *neq* ‹*lt* ∈ *set* $\mathcal{C}$› *clause*
unfolding *reduced-dnf-def reduce-clause-def* **by** *fastforce*
with $\mathcal{B}$*-sat-clause* **have** *interp* I_{sa} $\mathcal{B}$ *?lt* **by** *fastforce*
with *diff* **show** *?thesis* **by** *simp*

next
case (*single x y*)

let *?lt* = *AT* (*Var* (*Solo x*) =$_s$ *Single* (*Var* (*Solo y*)))
from *single* **have** *?lt* ∈ *reduce-literal lt*
using ‹*lt* ∈ *set* $\mathcal{C}$› **by** *simp*
then have *?lt* ∈ *clause*
using *neq* ‹*lt* ∈ *set* $\mathcal{C}$› *clause*
unfolding *reduced-dnf-def reduce-clause-def* **by** *fastforce*
with $\mathcal{B}$*-sat-clause* **have** *interp* I_{sa} $\mathcal{B}$ *?lt* **by** *fastforce*
with *single* **show** *?thesis* **by** *simp*

next
case (*app x f y*)
with ‹*lt* ∈ *set* $\mathcal{C}$› **have** *f* ∈ *F* **unfolding** *F-def* **by** *force*
from $\mathcal{B}$*-sat-clause clause eval-v*
have $\mathcal{B}$*-v*: ($\mathcal{B}$ ∘ *VennRegion*) α = *proper-Venn-region* α **if** $\alpha \in P^+$ *V* **for** α
unfolding *reduced-dnf-def*
using *proper-Venn-region.simps that* **by** *force*
from $\mathcal{B}$*-sat-clause clause eval-w*
have $\mathcal{B}$*-w*: $\bigsqcup$*HF* ((*B* ∘ *VennRegion*) ‘ *l*) = $\bigsqcup$*HF* ((*B* ∘ *VennRegion*) ‘ *m*) ⟶ $\mathcal{B}$ w_{fl} = $\mathcal{B}$ w_{fm}
if *l* ⊆ P^+ *V m* ⊆ P^+ *V f* ∈ *F* **for** *l m f*
by (*meson in-mono introduce-UnionOfVennRegions-subset-reduced-fms introduce-w-subset-reduced-fms that*)

from *app* ‹*lt* ∈ *set* $\mathcal{C}$› **have** *y* ∈ *V* **using** *V-def* **by** *fastforce*
with *variable-as-composition-of-proper-Venn-regions*
have $\bigsqcup$*HF* (*proper-Venn-region* ‘ $\mathcal{L}$ *V y*) = $discretize_v$ $\mathcal{B}$ *y* **by** *blast*
with Λ*-discr* $\mathcal{L}$*-subset-P-plus* $\mathcal{B}$*-v*
have $\bigsqcup$*HF* ((*B* ∘ *VennRegion*) ‘ $\mathcal{L}$ *V y*) = $\bigsqcup$*HF* ((*B* ∘ *VennRegion*) ‘ Λ ($discretize_v$ $\mathcal{B}$ *y*))
by (*smt* (*verit*, *best*) *HUnion-eq subset-eq*)
with $\mathcal{B}$*-w* **have** $\mathcal{B}$*-w-eq*: $\mathcal{B}$ $w_{f\mathcal{L}}$ *V y* = $\mathcal{B}$ $w_{f\Lambda}$ ($discretize_v$ $\mathcal{B}$ *y*)
using $\mathcal{L}$*-subset-P-plus* Λ*-subset-V* ‹*f* ∈ *F*› *finite-V* **by** *meson*

let *?lt* = *AT* (*Var* (*Solo x*) =$_s$ *Var* $w_{f\mathcal{L}}$ *V y*)
from *app* **have** *?lt* ∈ *reduce-literal lt*
using ‹*lt* ∈ *set* $\mathcal{C}$› **by** *simp*
then have *?lt* ∈ *clause*
using *neq* ‹*lt* ∈ *set* $\mathcal{C}$› *clause*
unfolding *reduced-dnf-def reduce-clause-def* **by** *fastforce*
with $\mathcal{B}$*-sat-clause* **have** *interp* I_{sa} $\mathcal{B}$ *?lt* **by** *fastforce*
then have $\mathcal{B}$ (*Solo x*) = $\mathcal{B}$ $w_{f\mathcal{L}}$ *V y* **by** *simp*
with $\mathcal{B}$*-w-eq* **have** $\mathcal{B}$ (*Solo x*) = $\mathcal{B}$ $w_{f\Lambda}$ ($discretize_v$ $\mathcal{B}$ *y*) **by** *argo*
then have $\mathcal{B}$ (*Solo x*) = ($discretize_f$ $\mathcal{B}$ *f*) ($discretize_v$ $\mathcal{B}$ *y*) **by** *simp*
then have $discretize_v$ $\mathcal{B}$ *x* = ($discretize_f$ $\mathcal{B}$ *f*) ($discretize_v$ $\mathcal{B}$ *y*) **by** *simp*
with *app* **show** *?thesis* **by** *simp*
qed
qed

lemma $\mathcal{C}$*-sat*: I_{cl} ($discretize_v$ $\mathcal{B}$) ($discretize_f$ $\mathcal{B}$) $\mathcal{C}$

using *all-literal-sat* **by** *blast*

end

lemma (**in** *normalized-MLSSmf-clause*) *MLSSmf-to-MLSS-completeness*:
assumes *is-model-for-reduced-dnf M*
shows $\exists\, M_v\ M_f.\ I_{cl}\ M_v\ M_f\ \mathcal{C}$
proof −
from *assms singleton-model-for-reduced-MLSS-clause* **obtain** $\mathcal{M}$ **where**
$\mathcal{M}$-*singleton*: $\forall\, \alpha \in P^+\ V.\ hcard\ (\mathcal{M}\ (v_\alpha)) \leq 1$ **and**
$\mathcal{M}$-*model*: *is-model-for-reduced-dnf* $\mathcal{M}$
using *normalized-MLSSmf-clause-axioms V-def* **by** *blast*
then obtain *clause* **where** *clause* ∈ *reduced-dnf* $\forall\, lt \in clause.\ interp\ I_{sa}\ \mathcal{M}\ lt$
unfolding *is-model-for-reduced-dnf-def* **by** *blast*
with *normalized-clause-contains-all-v-α* **have** *v-α-in-vars*:
$\forall\, \alpha{\in}P^+\ V.\ v_\alpha \in \bigcup\ (vars\ `\ clause)$
by *blast*

from $\mathcal{M}$-*singleton* **have** *assigned-set-card-0-or-1*:
$\forall\, \alpha \in P^+\ V.\ hcard\ (\mathcal{M}\ (v_\alpha)) = 0 \vee hcard\ (\mathcal{M}\ (v_\alpha)) = 1$
using *antisym-conv2* **by** *blast*

let $?\Lambda = \lambda a.\ \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap a \neq 0\}$

have Λ-*subset-V*: $?\Lambda\ x \subseteq P^+\ V$ **for** *x*
by *fast*

have Λ-*preserves-zero*: $?\Lambda\ 0 = \{\}$ **by** *blast*

have Λ-*inc*: $a \leq b \Longrightarrow ?\Lambda\ a \subseteq ?\Lambda\ b$ **for** *a b*
by (*smt* (*verit*) *Collect-mono hinter-hempty-right inf.absorb-iff1 inf-left-commute*)

have Λ-*add*: $?\Lambda\ (a \sqcup b) = ?\Lambda\ a \cup ?\Lambda\ b$ **for** *a b*
proof (*standard*; *standard*)
fix α **assume** α: $\alpha \in \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap (a \sqcup b) \neq 0\}$
then have $\alpha \in P^+\ V\ \mathcal{M}\ v_\alpha \sqcap (a \sqcup b) \neq 0$ **by** *blast+*
then have $\mathcal{M}\ v_\alpha \sqcap a \neq 0 \vee \mathcal{M}\ v_\alpha \sqcap b \neq 0$
by (*metis hunion-hempty-right inf-sup-distrib1*)
then show $\alpha \in \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap a \neq 0\} \cup \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap b \neq 0\}$
using α **by** *blast*
next
fix α **assume** $\alpha \in \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap a \neq 0\} \cup \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap b \neq 0\}$
then have $\alpha \in P^+\ V\ \mathcal{M}\ v_\alpha \sqcap a \neq 0 \vee \mathcal{M}\ v_\alpha \sqcap b \neq 0$
by *blast+*
then have $\mathcal{M}\ v_\alpha \sqcap (a \sqcup b) \neq 0$
by (*metis hinter-hempty-right hunion-hempty-left inf-sup-absorb inf-sup-distrib1*)
then show $\alpha \in \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap (a \sqcup b) \neq 0\}$
using ‹$\alpha \in P^+\ V$› **by** *blast*

qed

have Λ-*mul*: *?*Λ $(a \sqcap b) =$ *?*Λ $a \cap$ *?*Λ b **for** a b
proof (*standard*; *standard*)
fix α **assume** α: $\alpha \in \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap (a \sqcap b) \neq 0\}$
then have $\alpha \in P^+\ V\ \mathcal{M}\ v_\alpha \sqcap (a \sqcap b) \neq 0$ **by** *blast+*
then have $\mathcal{M}\ v_\alpha \sqcap a \neq 0 \wedge \mathcal{M}\ v_\alpha \sqcap b \neq 0$
by (*metis hinter-hempty-left inf-assoc inf-left-commute*)
then show $\alpha \in \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap a \neq 0\} \cap \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap b \neq 0\}$
using α **by** *blast*
next
fix α **assume** $\alpha \in \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap a \neq 0\} \cap \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap b \neq 0\}$
then have $\alpha \in P^+\ V\ \mathcal{M}\ v_\alpha \sqcap a \neq 0\ \mathcal{M}\ v_\alpha \sqcap b \neq 0$
by *blast+*
then have $\mathcal{M}\ v_\alpha \neq 0$ **by** *force*
then have *hcard* $(\mathcal{M}\ v_\alpha) \neq 0$ **using** *hcard-0E* **by** *blast*
then have *hcard* $(\mathcal{M}\ v_\alpha) = 1$
using *assigned-set-card-0-or-1 v-α-in-vars* ‹$\alpha \in P^+\ V$›
by *fastforce*
then obtain c **where** $\mathcal{M}\ v_\alpha = 0 \triangleleft c$
using *hcard-1E* **by** *blast*
moreover
from ‹$\mathcal{M}\ v_\alpha = 0 \triangleleft c$› ‹$\mathcal{M}\ v_\alpha \sqcap a \neq 0$›
have $\mathcal{M}\ v_\alpha \sqcap a = 0 \triangleleft c$ **by** *auto*
moreover
from ‹$\mathcal{M}\ v_\alpha = 0 \triangleleft c$› ‹$\mathcal{M}\ v_\alpha \sqcap b \neq 0$›
have $\mathcal{M}\ v_\alpha \sqcap b = 0 \triangleleft c$ **by** *auto*
ultimately
have $\mathcal{M}\ v_\alpha \sqcap (a \sqcap b) = 0 \triangleleft c$
by (*simp add*: *inf-commute inf-left-commute*)
then have $\mathcal{M}\ v_\alpha \sqcap (a \sqcap b) \neq 0$ **by** *simp*
then show $\alpha \in \{\alpha \in P^+\ V.\ \mathcal{M}\ v_\alpha \sqcap (a \sqcap b) \neq 0\}$
using ‹$\alpha \in P^+\ V$› **by** *blast*
qed

have $l \subseteq P^+\ V \Longrightarrow$
$a = \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l) \Longrightarrow a \leq \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `$
(*?*Λ a)) **for** l a
proof
fix c **assume** *l-a-c*: $l \subseteq P^+\ V\ a = \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l)\ c \in a$
then obtain α **where** $\alpha \in l\ c \in \mathcal{M}\ v_\alpha$ **by** *auto*
then have $\alpha \in$ *?*Λ a **using** *l-a-c* **by** *blast*
then have $\mathcal{M}\ v_\alpha \in (\mathcal{M} \circ VennRegion)\ `$ (*?*Λ a) **by** *simp*
then have $\mathcal{M}\ v_\alpha \in HF\ ((\mathcal{M} \circ VennRegion)\ `$ (*?*Λ a)) **by** *fastforce*
with ‹$c \in \mathcal{M}\ v_\alpha$› **show** $c \in \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `$ (*?*Λ a)) **by** *blast*
qed
moreover
have $l \subseteq P^+\ V \Longrightarrow$

$a = \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l) \Longrightarrow \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ (?\Lambda\ a)) \leq a$ **for** $l\ a$

proof −

assume $l \subseteq P^+\ V$ **and** a: $a = \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l)$

then have *finite l*

by (*simp add*: *finite-V finite-subset*)

have $?\Lambda\ a \subseteq l$

proof

fix α **assume** $\alpha \in ?\Lambda\ a$

then obtain c **where** $c \in \mathcal{M}\ v_\alpha \sqcap a$ **by** *blast*

then have $c \in \mathcal{M}\ v_\alpha\ c \in a$ **by** *blast+*

then obtain β **where** $\beta \in l\ c \in \mathcal{M}\ v_\beta$ **using** a **by** *force*

interpret *proper-Venn-regions V* $\mathcal{M} \circ Solo$

using *finite-V* **by** *unfold-locales*

from ‹$\alpha \in ?\Lambda\ a$› **have** $\alpha \in P^+\ V$ **by** *auto*

moreover

from ‹$l \subseteq P^+\ V$› ‹$\beta \in l$› **have** $\beta \in P^+\ V$ **by** *auto*

moreover

from ‹$c \in \mathcal{M}\ v_\alpha$› **have** $c \in$ *proper-Venn-region* α

using *eval-v* ‹$\alpha \in P^+\ V$› $\mathcal{M}$*-model*

unfolding *is-model-for-reduced-dnf-def reduced-dnf-def*

by *fastforce*

moreover

from ‹$c \in \mathcal{M}\ v_\beta$› **have** $c \in$ *proper-Venn-region* β

using *eval-v* ‹$\beta \in P^+\ V$› $\mathcal{M}$*-model*

unfolding *is-model-for-reduced-dnf-def reduced-dnf-def*

by *fastforce*

ultimately

have $\alpha = \beta$

using *finite-V proper-Venn-region-strongly-injective* **by** *auto*

with ‹$\beta \in l$› **show** $\alpha \in l$ **by** *simp*

qed

then have $(\mathcal{M} \circ VennRegion)\ `\ ?\Lambda\ a \subseteq (\mathcal{M} \circ VennRegion)\ `\ l$ **by** *blast*

moreover

from ‹*finite l*› **have** *finite* $((\mathcal{M} \circ VennRegion)\ `\ l)$ **by** *blast*

ultimately

have $\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ ?\Lambda\ a) \leq \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l)$

by (*metis* (*no-types*, *lifting*) *HUnion-hunion finite-subset sup.orderE sup.orderI union-hunion*)

then show $\bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ (?\Lambda\ a)) \leq a$

using a **by** *blast*

qed

ultimately

have Λ*-discr*: $l \subseteq P^+\ V \Longrightarrow a = \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ l) \Longrightarrow a = \bigsqcup HF\ ((\mathcal{M} \circ VennRegion)\ `\ (?\Lambda\ a))$ **for** $l\ a$

by (*simp add*: *inf.absorb-iff1 inf-commute*)

interpret Λ-*plus*: *MLSSmf-to-MLSS-complete* $\mathcal{C}$ $\mathcal{M}$ *?*Λ
using *assms* $\mathcal{M}$-*singleton* $\mathcal{M}$-*model*
Λ-*subset-V* Λ-*preserves-zero* Λ-*inc* Λ-*add* Λ-*mul* Λ-*discr*
by *unfold-locales*

show *?thesis*
using Λ-*plus*.$\mathcal{C}$-*sat* **by** *fast*
qed

end
theory *MLSSmf-to-MLSS-Correctness*
imports *MLSSmf-to-MLSS-Soundness MLSSmf-to-MLSS-Completeness*
begin

fun *reduce* :: ('*v*, '*f*) *MLSSmf-clause* $\Rightarrow$ ('*v*, '*f*) *Composite pset-fm set set* **where**
reduce $\mathcal{C}$ = *normalized-MLSSmf-clause.reduced-dnf* $\mathcal{C}$

fun *interp-DNF* :: (('*v*, '*f*) *Composite* $\Rightarrow$ *hf*) $\Rightarrow$ ('*v*, '*f*) *Composite pset-fm set set* $\Rightarrow$ *bool* **where**
interp-DNF $\mathcal{M}$ *clauses* $\longleftrightarrow$ ($\exists$ *clause* $\in$ *clauses*. $\forall$ *lt* $\in$ *clause*. *interp* I_{sa} $\mathcal{M}$ *lt*)

corollary *MLSSmf-to-MLSS-correct*:
assumes *norm-clause* $\mathcal{C}$
shows ($\exists$ M_v M_f. I_{cl} M_v M_f $\mathcal{C}$) $\longleftrightarrow$ ($\exists$ $\mathcal{M}$. *interp-DNF* $\mathcal{M}$ (*reduce* $\mathcal{C}$))
proof
from *assms* **interpret** *normalized-MLSSmf-clause* $\mathcal{C}$ **by** *unfold-locales*
assume $\exists$ M_v M_f. I_{cl} M_v M_f $\mathcal{C}$
with *MLSSmf-to-MLSS-soundness* **obtain** $\mathcal{M}$ **where** *is-model-for-reduced-dnf* $\mathcal{M}$
using *assms* **by** *blast*
then have *interp-DNF* $\mathcal{M}$ (*reduce* $\mathcal{C}$) **unfolding** *is-model-for-reduced-dnf-def* **by** *simp*
then show $\exists$ $\mathcal{M}$. *interp-DNF* $\mathcal{M}$ (*reduce* $\mathcal{C}$) **by** *blast*
next
from *assms* **interpret** *normalized-MLSSmf-clause* $\mathcal{C}$ **by** *unfold-locales*
assume $\exists$ $\mathcal{M}$. *interp-DNF* $\mathcal{M}$ (*reduce* $\mathcal{C}$)
then obtain $\mathcal{M}$ **where** *interp-DNF* $\mathcal{M}$ (*reduce* $\mathcal{C}$) **by** *blast*
then have *is-model-for-reduced-dnf* $\mathcal{M}$ **unfolding** *is-model-for-reduced-dnf-def* **by** *simp*
with *MLSSmf-to-MLSS-completeness* **show** $\exists$ M_v M_f. I_{cl} M_v M_f $\mathcal{C}$ **by** *blast*
qed

end

References

[1] Domenico Cantone, Jacob T. Schwartz, and Calogero G. Zarba. A decision procedure for a sublanguage of set theory involving monotone additive and multiplicative functions, ii. the multi-level case. *Le Matematiche; Vol 60, No 1 (2005); 133-162*, 60, 01 2006.

[2] Lukas Stevens. Mlss decision procedure. *Archive of Formal Proofs*, May 2023. ISSN 2150-914x. https://isa-afp.org/entries/MLSS_Decision_Proc.html, Formal proof development.