A Modular Splitting Framework for Saturation Theorem Proving

Ghilain Bergeron, Florent Krasnopol and Sophie Tourret

September 9, 2026

Abstract

We formalize in Isabelle/HOL a framework for splitting [2], a theorem proving technique that extends saturation-based calculi with branching abilities. The framework preserves the completeness of the original calculus. We focus here on the simplest splitting model described in details in the first three sections of "Unifying Splitting" by Gabriel Ebner, Jasmin Blanchette and Sophie Tourret [3] and provide an extension of the ordered resolution calculus with a variant of splitting called Lightweight AVATAR. A paper describing the present formalization has been accepted at ITP'25 [1].

Contents

1 Limsup of Lazy Lists **2**
- 1.1 Library . . . 2
- 1.2 Infimum . . . 2
- 1.3 Infimum up-to . . . 3
- 1.4 Limsup . . . 5
- 1.5 Limsup up-to . . . 7

2 Disjunctive Consequence Relations **11**

3 Extension to Negated Formulas **30**

4 Calculi and Derivations **37**

5 Annotated Formulas and Consequence Relations **43**

6 Lifting Calculi to Add Annotations **46**
- 6.1 Local saturation . . . 95

7 Lifting to Non-ground Calculi **96**
7.1 Standard Lifting . . . 97
7.2 Strong Standard Lifting . . . 98
7.3 Lifting with a Family of Tiebreaker Orderings . . . 99
7.4 Lifting with a Family of Redundancy Criteria . . . 108

8 Core splitting calculus **112**
8.1 The inference rules . . . 113
8.2 The redundancy criterion . . . 118
8.3 Standard completeness . . . 130
8.4 Strong completeness . . . 136

9 Extensions: Inferences and simplifications **139**
9.1 Simplifications . . . 139
9.1.1 The Split Rule . . . 140
9.1.2 The Collect Rule . . . 150
9.1.3 The Trim Rule . . . 154
9.2 Extra Inferences . . . 160
9.2.1 The StrongUnsat Rule . . . 160
9.2.2 The Tauto Rule . . . 162
9.2.3 The Approx Rule . . . 164
9.3 Combining all simplifications and optional inferences . . . 166
9.4 The BinSplit Simplification Rule . . . 168
9.5 Ordered Resolution with a Disjunctive Consequence Relation 174
9.6 Lightweight Avatar without BinSplit . . . 184
9.7 Lightweight Avatar . . . 186

1 Limsup of Lazy Lists

theory *Lazy-List-Limsup*
imports *Coinductive.Coinductive-List*
begin

1.1 Library

lemma *less-llength-ltake*: $i < llength\ (ltake\ k\ Xs) \longleftrightarrow i < k \wedge i < llength\ Xs$
by *simp*

1.2 Infimum

definition *Inf-llist* :: ‹*'a set llist* ⇒ *'a set*› **where**
‹*Inf-llist Xs* = $(\bigcap i \in \{i.\ enat\ i < llength\ Xs\}.\ lnth\ Xs\ i)$›

lemma *Inf-llist-subset-lnth*: ‹*enat i* < *llength Xs* ⟹ *Inf-llist Xs* ⊆ *lnth Xs i*›
unfolding *Inf-llist-def* **by** *fast*

lemma *Inf-llist-imp-exists-index*:
assumes ‹¬ *lnull Xs*›
shows ‹*x* ∈ *Inf-llist Xs* ⟹ ∃ *i*. *enat i* < *llength Xs* ∧ *x* ∈ *lnth Xs i*›
unfolding *Inf-llist-def* **using** *assms*
by (*metis INT-E i0-less llength-eq-0 mem-Collect-eq zero-enat-def*)

lemma *Inf-llist-imp-all-index*: ‹*x* ∈ *Inf-llist Xs* ⟹ ∀ *i*. *enat i* < *llength Xs* ⟶ *x* ∈ *lnth Xs i*›
unfolding *Inf-llist-def* **by** *blast*

lemma *all-index-imp-Inf-llist*: ‹∀ *i*. *enat i* < *llength Xs* ⟶ *x* ∈ *lnth Xs i* ⟹ *x* ∈ *Inf-llist Xs*›
unfolding *Inf-llist-def* **by** *auto*

lemma *Inf-llist-LNil*[*simp*]: ‹*Inf-llist LNil* = *UNIV*›
unfolding *Inf-llist-def* **by** *auto*

lemma *Inf-llist-LCons*[*simp*]: ‹*Inf-llist* (*LCons X Xs*) = *X* ∩ *Inf-llist Xs*›
unfolding *Inf-llist-def*
proof (*intro subset-antisym subsetI*)
fix *x*
assume ‹*x* ∈ (⋂ *i* ∈ {*i*. *enat i* < *llength* (*LCons X Xs*)}. *lnth* (*LCons X Xs*) *i*)›
then have ‹*enat i* < *llength* (*LCons X Xs*) ⟶ *x* ∈ *lnth* (*LCons X Xs*) *i*› **for** *i*
by *blast*
then have ‹*x* ∈ *X* ∧ (∀ *i*. *enat i* < *llength Xs* ⟶ *x* ∈ *lnth Xs i*)›
by (*metis Suc-ile-eq iless-Suc-eq llength-LCons lnth-0 lnth-Suc-LCons zero-enat-def zero-le*)
then show ‹*x* ∈ *X* ∩ (⋂ *i* ∈ {*i*. *enat i* < *llength Xs*}. *lnth Xs i*)›
by *blast*
next
fix *x*
assume ‹*x* ∈ *X* ∩ ⋂ (*lnth Xs* ‘ {*i*. *enat i* < *llength Xs*})›
then have ‹*x* ∈ *X* ∧ (∀ *i*. *enat i* < *llength Xs* ⟶ *x* ∈ *lnth Xs i*)›
by *simp*
then have ‹∀ *i*. *enat i* ≤ *llength Xs* ⟶ *x* ∈ *lnth* (*LCons X Xs*) *i*›
by (*metis Suc-diff-1 Suc-ile-eq lnth-0 lnth-Suc-LCons neq0-conv*)
then show ‹*x* ∈ ⋂ (*lnth* (*LCons X Xs*) ‘ {*i*. *enat i* < *llength* (*LCons X Xs*)})›
by *simp*
qed

lemma *lhd-subset-Inf-llist*: ‹¬ *lnull Xs* ⟹ *Inf-llist Xs* ⊆ *lhd Xs*›
by (*cases Xs*) *simp-all*

1.3 Infimum up-to

definition *Inf-upto-llist* :: ‹'*a set llist* ⇒ *enat* ⇒ '*a set*› **where**
‹*Inf-upto-llist Xs j* = (⋂ *i* ∈ {*i*. *enat i* < *llength Xs* ∧ *enat i* ≤ *j*}. *lnth Xs i*)›

lemma *Inf-upto-llist-eq-Inf-llist-ltake*: ‹*Inf-upto-llist Xs j* = *Inf-llist* (*ltake* (*eSuc*

j) *Xs*)›
unfolding *Inf-upto-llist-def Inf-llist-def*
by (*smt* (*verit*, *best*) *Collect-cong Sup.SUP-cong iless-Suc-eq less-llength-ltake lnth-ltake*
mem-Collect-eq)

lemma *Inf-upto-llist-enat-0*[*simp*]:
‹*Inf-upto-llist Xs* (*enat 0*) = (*if lnull Xs then UNIV else lhd Xs*)›
unfolding *Inf-upto-llist-def image-def*
by (*cases* ‹*lnull Xs*›) (*auto simp*: *lhd-conv-lnth enat-0 enat-0-iff*)

lemma *Inf-upto-llist-Suc*[*simp*]:
‹*Inf-upto-llist Xs* (*enat* (*Suc j*)) =
Inf-upto-llist Xs (*enat j*) ∩ (*if enat* (*Suc j*) < *llength Xs then lnth Xs* (*Suc j*) *else UNIV*)›
unfolding *Inf-upto-llist-def image-def* **by** (*auto intro*: *le-SucI elim*: *le-SucE*)

lemma *Inf-upto-llist-infinity*[*simp*]: ‹*Inf-upto-llist Xs* ∞ = *Inf-llist Xs*›
unfolding *Inf-upto-llist-def Inf-llist-def* **by** *simp*

lemma *Inf-upto-llist-0*[*simp*]: ‹*Inf-upto-llist Xs 0* = (*if lnull Xs then UNIV else lhd Xs*)›
unfolding *zero-enat-def* **by** (*rule Inf-upto-llist-enat-0*)

lemma *Inf-upto-llist-eSuc*[*simp*]:
‹*Inf-upto-llist Xs* (*eSuc j*) =
(*case j of*
enat k ⇒ *Inf-upto-llist Xs* (*enat* (*Suc k*))
| ∞ ⇒ *Inf-llist Xs*)›
by (*auto simp*: *eSuc-enat split*: *enat.split*)

lemma *Inf-upto-llist-anti*[*simp*]: ‹*j* ≤ *k* ⟹ *Inf-upto-llist Xs k* ⊆ *Inf-upto-llist Xs j*›
unfolding *Inf-upto-llist-def* **by** *auto*

lemma *Inf-llist-subset-Inf-upto-llist*: ‹*Inf-llist Xs* ⊆ *Inf-upto-llist Xs j*›
unfolding *Inf-llist-def Inf-upto-llist-def* **by** *auto*

lemma *elem-Inf-llist-imp-Inf-upto-llist*:
‹*x* ∈ *Inf-llist Xs* ⟹ *x* ∈ *Inf-upto-llist Xs* (*enat j*)›
unfolding *Inf-llist-def Inf-upto-llist-def* **by** *blast*

lemma *Inf-upto-llist-subset-lnth*: ‹*j* < *llength Xs* ⟹ *Inf-upto-llist Xs j* ⊆ *lnth Xs j*›
unfolding *Inf-upto-llist-def* **by** *auto*

lemma *Inf-llist-imp-Inf-upto-llist*:
assumes ‹*X* ⊆ *Inf-llist Xs*›
shows ‹*X* ⊆ *Inf-upto-llist Xs* (*enat k*)›

using *assms elem-Inf-llist-imp-Inf-upto-llist* **by** *fast*

1.4 Limsup

definition *Limsup-llist* :: ‹*'a set llist* ⇒ *'a set*› **where**
‹*Limsup-llist Xs* =
(⋂ *i* ∈ {*i*. *enat i* < *llength Xs*}. ⋃ *j* ∈ {*j*. *i* ≤ *j* ∧ *enat j* < *llength Xs*}. *lnth Xs j*)›

lemma *Limsup-llist-LNil*[*simp*]: ‹*Limsup-llist LNil* = *UNIV*›
unfolding *Limsup-llist-def* **by** *simp*

lemma *Limsup-llist-LCons*:
‹*Limsup-llist* (*LCons X Xs*) = (*if lnull Xs then X else Limsup-llist Xs*)› (**is** ‹*?lhs* = *?rhs*›)
proof (*cases lnull Xs*)
case *nnull*: *False*
show *?thesis*
proof
{
fix *x*
assume ‹∀ *i*. *enat i* ≤ *llength Xs* ⟶ (∃ *j* ≥ *i*. *enat j* ≤ *llength Xs* ∧ *x* ∈ *lnth* (*LCons X Xs*) *j*)›
then have ‹∀ *i*. *enat i* < *llength Xs* ⟶ (∃ *j* ≥ *i*. *enat j* < *llength Xs* ∧ *x* ∈ *lnth Xs j*)›
by (*metis Suc-ile-eq Suc-le-D Suc-le-mono lnth-Suc-LCons*)
}
then show ‹*?lhs* ⊆ *?rhs*›
by (*auto simp*: *Limsup-llist-def nnull*)

{
fix *x*
assume ‹∀ *i*. *enat i* < *llength Xs* ⟶ (∃ *j* ≥ *i*. *enat j* < *llength Xs* ∧ *x* ∈ *lnth Xs j*)›
then have ‹∀ *i*. *enat i* ≤ *llength Xs* ⟶
(∃ *j* ≥ *i*. *enat j* ≤ *llength Xs* ∧ *x* ∈ *lnth* (*LCons X Xs*) *j*)›
by (*metis Suc-ile-eq Suc-le-D iless-Suc-eq llength-LCons lnth-Suc-LCons nat-le-linear*
nnull not-less-eq-eq not-lnull-conv zero-enat-def zero-le)
}
then show *?rhs* ⊆ *?lhs*
by (*auto simp*: *Limsup-llist-def nnull*)
qed
qed (*simp add*: *Limsup-llist-def enat-0-iff*(*1*))

lemma *lfinite-Limsup-llist*: ‹*lfinite Xs* ⟹ *Limsup-llist Xs* = (*if lnull Xs then UNIV else llast Xs*)›
proof (*induction rule*: *lfinite-induct*)
case (*LCons xs*)

then obtain *y ys* **where** *xs*: ‹*xs = LCons y ys*›
by (*meson not-lnull-conv*)
show *?case*
unfolding *xs* **by** (*simp add*: *Limsup-llist-LCons LCons.IH*[*unfolded xs*, *simplified*] *llast-LCons*)
qed (*simp add*: *Limsup-llist-def*)

lemma *Limsup-llist-ltl*: ‹¬ *lnull* (*ltl Xs*) ⟹ *Limsup-llist Xs* = *Limsup-llist* (*ltl Xs*)›
by (*metis Limsup-llist-LCons lhd-LCons-ltl lnull-ltlI*)

lemma *Inf-llist-subset-Limsup-llist*: ‹*Inf-llist Xs* ⊆ *Limsup-llist Xs*›
unfolding *Limsup-llist-def Inf-llist-def* **by** *fast*

lemma *image-Limsup-llist-subset*: ‹*f* ‘ *Limsup-llist Ns* ⊆ *Limsup-llist* (*lmap* ((‘) *f*) *Ns*)›
unfolding *Limsup-llist-def* **by** *fastforce*

lemma *Limsup-llist-imp-exists-index*:
assumes ‹¬ *lnull Xs*›
shows ‹*x* ∈ *Limsup-llist Xs* ⟹ ∃ *i*. *enat i* < *llength Xs* ∧ *x* ∈ *lnth Xs i*›
unfolding *Limsup-llist-def* **using** *assms*
by *simp* (*metis i0-less llength-eq-0 zero-enat-def*)

lemma *finite-subset-Limsup-llist-imp-exists-index*:
assumes
nnil: ‹¬ *lnull Xs*› **and**
fin: ‹*finite X*› **and**
in-sup: ‹*X* ⊆ *Limsup-llist Xs*›
shows ‹∀ *i*. *enat i* < *llength Xs* ⟶ *X* ⊆ (⋃ *j* ∈ {*j*. *i* ≤ *j* ∧ *enat j* < *llength Xs*}. *lnth Xs j*)›
using *assms* **unfolding** *Limsup-llist-def* **by** *blast*

lemma *Limsup-llist-lmap-image*:
assumes *f-inj*: ‹*inj-on f* (*Inf-llist* (*lmap g xs*))›
shows ‹*Limsup-llist* (*lmap* (λ*x*. *f* ‘ *g x*) *xs*) = *f* ‘ *Limsup-llist* (*lmap g xs*)› (**is** ‹*?lhs* = *?rhs*›)
oops

lemma *Limsup-llist-lmap-union*:
assumes ‹∀ *x* ∈ *lset xs*. ∀ *y* ∈ *lset xs*. *g x* ∩ *h y* = {}›
shows ‹*Limsup-llist* (*lmap* (λ*x*. *g x* ∪ *h x*) *xs*) =
Limsup-llist (*lmap g xs*) ∪ *Limsup-llist* (*lmap h xs*)› (**is** ‹*?lhs* = *?rhs*›)
proof (*intro equalityI subsetI*)
fix *x*
assume *x-in*: ‹*x* ∈ *?lhs*›
then have ‹∀ *i*. *enat i* < *llength xs* ⟶ (∃ *j* ≥ *i*. *enat j* < *llength xs* ∧
(*x* ∈ *g* (*lnth xs j*) ∨ *x* ∈ *h* (*lnth xs j*)))›
unfolding *Limsup-llist-def* **by** *auto*

then have ‹$(\forall i'.\ enat\ i' < llength\ xs \longrightarrow (\exists j \geq i'.\ enat\ j < llength\ xs \wedge x \in g\ (lnth\ xs\ j)))$
$\vee\ (\forall i'.\ enat\ i' < llength\ xs \longrightarrow (\exists j \geq i'.\ enat\ j < llength\ xs \wedge x \in h\ (lnth\ xs\ j)))$›
using *assms*[*unfolded disjoint-iff-not-equal*] **by** (*metis in-lset-conv-lnth*)
then show ‹$x \in ?rhs$›
unfolding *Limsup-llist-def* **by** *simp*
next
fix x
show ‹$x \in ?rhs \Longrightarrow x \in ?lhs$›
using *assms* **unfolding** *Limsup-llist-def* **by** *auto*
qed

lemma *Limsup-set-filter-commute*:
assumes ‹$\neg$ *lnull Xs*›
shows ‹*Limsup-llist* (*lmap* ($\lambda X.\ \{x \in X.\ p\ x\}$) *Xs*) = $\{x \in$ *Limsup-llist Xs*. *p* $x\}$› (**is** ‹*?lhs* = *?rhs*›)
proof (*intro equalityI subsetI*)
fix x
assume ‹$x \in ?lhs$›
then show ‹$x \in ?rhs$›
unfolding *Limsup-llist-def*
by *simp* (*metis assms i0-less llength-eq-0 zero-enat-def*)
qed (*simp add*: *Limsup-llist-def*)

1.5 Limsup up-to

definition *Limsup-upto-llist* :: ‹$'a$ *set llist* $\Rightarrow$ *enat* $\Rightarrow$ $'a$ *set*› **where**
‹*Limsup-upto-llist Xs k* =
($\bigcap i \in \{i.\ enat\ i < llength\ Xs \wedge enat\ i \leq k\}$.
$\bigcup j \in \{j.\ i \leq j \wedge enat\ j < llength\ Xs \wedge enat\ j \leq k\}$. *lnth Xs j*)›

lemma *Limsup-upto-llist-eq-Limsup-llist-ltake*:
‹*Limsup-upto-llist Xs j* = *Limsup-llist* (*ltake* (*eSuc j*) *Xs*)›
unfolding *Limsup-upto-llist-def Limsup-llist-def*
by (*smt* (*z3*) *Collect-cong Sup.SUP-cong iless-Suc-eq lnth-ltake less-llength-ltake mem-Collect-eq*)

lemma *Limsup-upto-llist-enat*[*simp*]:
‹*Limsup-upto-llist Xs* (*enat k*) =
(*if enat k* < *llength Xs then lnth Xs k else if lnull Xs then UNIV else llast Xs*)›
proof (*cases* ‹*enat k* < *llength Xs*›)
case *True*
then show *?thesis*
unfolding *Limsup-upto-llist-def* **by** *force*
next
case *k-ge*: *False*
show *?thesis*
proof (*cases* ‹*lnull Xs*›)

case *nil*: *True*
then show *?thesis*
unfolding *Limsup-upto-llist-def* **by** *simp*
next
case *nnil*: *False*
then obtain *j* **where**
j: ‹*eSuc* (*enat j*) = *llength Xs*›
using *k-ge* **by** (*metis eSuc-enat-iff enat-ile le-less-linear lhd-LCons-ltl llength-LCons*)

have *fin*: ‹*lfinite Xs*›
using *k-ge not-lfinite-llength* **by** *fastforce*
have *le-k*: ‹*enat i* < *llength Xs* ∧ *i* ≤ *k* ⟷ *enat i* < *llength Xs*› **for** *i*
using *k-ge linear order-le-less-subst2* **by** *fastforce*
have ‹*Limsup-upto-llist Xs* (*enat k*) = *llast Xs*›
using *j nnil lfinite-Limsup-llist*[*OF fin*] *nnil*
unfolding *Limsup-upto-llist-def Limsup-llist-def* **using** *llast-conv-lnth*[*OF j*[*symmetric*]]
by (*simp add*: *le-k*)
then show *?thesis*
using *k-ge nnil* **by** *simp*
qed
qed

lemma *Limsup-upto-llist-infinity*[*simp*]: ‹*Limsup-upto-llist Xs* ∞ = *Limsup-llist Xs*›
unfolding *Limsup-upto-llist-def Limsup-llist-def* **by** *simp*

lemma *Limsup-upto-llist-0*[*simp*]:
‹*Limsup-upto-llist Xs 0* = (*if lnull Xs then UNIV else lhd Xs*)›
unfolding *Limsup-upto-llist-def image-def*
by (*simp add*: *enat-0*[*symmetric*]) (*simp add*: *enat-0 lnth-0-conv-lhd*)

lemma *Limsup-upto-llist-eSuc*[*simp*]:
‹*Limsup-upto-llist Xs* (*eSuc j*) =
(*case j of*
enat k ⇒ *Limsup-upto-llist Xs* (*enat* (*Suc k*))
| ∞ ⇒ *Limsup-llist Xs*)›
by (*auto simp*: *eSuc-enat split*: *enat.split*)

lemma *Liminf-upto-llist-imp-elem-Limsup-llist*:
assumes ‹∃ *i* < *llength Xs*. ∀ *j* ≥ *i*. *j* < *llength Xs* ⟶ *x* ∈ *Limsup-upto-llist Xs* (*enat j*)›
shows ‹*x* ∈ *Limsup-llist Xs*›
using *assms* **by** (*simp add*: *Limsup-llist-def Limsup-upto-llist-def*)
(*metis dual-order.strict-implies-order enat-iless enat-less-imp-le enat-ord-simps*(*1*) *not-less*)

end

theory *FSet-Extra*
imports *Main HOL−Library.FSet*
begin

This theory contains some additional lemmas regarding sets and finite sets, which were useful in the process of proving some lemmas in `Modular_Splitting_Calculus.thy` and `Lightweight_Avatar.thy`.

lemma *Union-of-singleton-is-setcompr*: ‹$(\bigcup\ x \in A.\ \{\ y.\ y = f\ x \wedge P\ x\ \}) = \{\ f\ x\ |\ x.\ x \in A \wedge P\ x\ \}$›
by *auto*

lemma *finite-because-singleton*: ‹$(\forall\ C1 \in S.\ \forall\ C2 \in S.\ C1 = C2) \longrightarrow$ *finite* S›
for S
by (*metis finite.simps is-singletonI' is-singleton-the-elem*)

lemma *finite-union-of-finite-is-finite*: ‹*finite* $E \Longrightarrow (\forall\ D \in E.$ *finite*$(\{f\ C\ |C.\ P\ C \wedge g\ C = D\})) \Longrightarrow$
finite$(\{f\ C\ |C.\ P\ C \wedge g\ C \in E\})$›
proof −
assume *finite-E*: ‹*finite* E› **and**
all-finite: ‹$\forall\ D \in E.$ *finite*$(\{f\ C\ |C.\ P\ C \wedge g\ C = D\})$›
have ‹*finite* $(\bigcup\{\{f\ C\ |C.\ P\ C \wedge g\ C = D\}\ |D.\ D{\in}E\})$›
using *finite-E all-finite finite-UN-I*
by (*simp add*: *setcompr-eq-image*)
moreover have ‹$\{f\ C\ |C.\ P\ C \wedge g\ C \in E\} \subseteq \bigcup\{\{f\ C\ |C.\ P\ C \wedge g\ C = D\}\ |D.\ D{\in}E\}$›
by *blast*
ultimately show *?thesis* **by** (*meson finite-subset*)
qed

definition *list-of-fset* :: *'a fset* ⇒ *'a list* **where**
‹*list-of-fset* A = (*SOME l. fset-of-list l* = A)›

lemma *fin-set-fset*: *finite* $A \Longrightarrow \exists\ Af.$ *fset* $Af = A$ **by** (*metis finite-list fset-of-list.rep-eq*)

lemma *fimage-snd-zip-is-snd* [*simp*]:
‹*length* x = *length* $y \Longrightarrow (\lambda(x,\ y).\ y)\ |`|$ *fset-of-list* (*zip* x y) = *fset-of-list* y›
proof −
assume *length-x-eq-length-y*: ‹*length* x = *length* y›
have ‹$(\lambda(x,\ y).\ y)\ |`|$ *fset-of-list* A = *fset-of-list* (*map* $(\lambda(x,\ y).\ y)$ A)› **for** A
by *auto*
then show *?thesis*
using *length-x-eq-length-y*
by (*smt* (*verit*, *ccfv-SIG*) *cond-case-prod-eta map-snd-zip snd-conv*)
qed

lemma *if-in-ffUnion-then-in-subset*: ‹$x\ |\in|$ *ffUnion* $A \Longrightarrow \exists\ a.\ a\ |\in|\ A \wedge x\ |\in|\ a$›
by (*induct* ‹A› *rule*: *fset-induct*, *fastforce*+)

lemma *fset-ffUnion-subset-iff-all-fsets-subset*: ‹*fset* (*ffUnion A*) ⊆ *B* ⟷ *fBall A* (λ *x*. *fset x* ⊆ *B*)›
proof (*intro fBallI subsetI iffI*)
  fix *a x*
  assume *ffUnion-A-subset-B*: ‹*fset* (*ffUnion A*) ⊆ *B*› **and**
      a-in-A: ‹*a* |∈| *A*› **and**
      x-in-fset-a: ‹*x* ∈ *fset a*›
  then have ‹*x* |∈| *a*›
    by *simp*
  then have ‹*x* |∈| *ffUnion A*›
    by (*metis a-in-A ffunion-insert funion-iff set-finsert*)
  then show ‹*x* ∈ *B*›
    using *ffUnion-A-subset-B* **by** *blast*
next
  fix *x*
  assume *all-in-A-subset-B*: ‹*fBall A* (λ *x*. *fset x* ⊆ *B*)› **and**
      ‹*x* ∈ *fset* (*ffUnion A*)›
  then have ‹*x* |∈| *ffUnion A*›
    by *simp*
  then obtain *a* **where** ‹*a* |∈| *A*› **and**
      x-in-a: ‹*x* |∈| *a*›
    by (*meson if-in-ffUnion-then-in-subset*)
  then have ‹*fset a* ⊆ *B*›
    using *all-in-A-subset-B*
    by *blast*
  then show ‹*x* ∈ *B*›
    using *x-in-a* **by** *blast*
qed

lemma *fBall-fset-of-list-iff-Ball-set*: ‹*fBall* (*fset-of-list A*) *P* ⟷ *Ball* (*set A*) *P*›
  by (*simp add*: *fset-of-list.rep-eq*)

lemma *wf-fsubset*: ‹*wfP* (|⊂|)›
proof −
  have ‹*wfP* (λ *A B*. *fcard A* < *fcard B*)›
    by (*simp add*: *wfp-if-convertible-to-nat*)
  then show ‹*wfP* (|⊂|)›
    by (*simp add*: *wfP-pfsubset*)
qed

lemma *non-zero-fcard-of-non-empty-set*: ‹*fcard A* > *0* ⟷ *A* ≠ {||}›
  by (*metis bot.not-eq-extremum fcard-fempty less-numeral-extra*(*3*) *pfsubset-fcard-mono*)

lemma *fimage-of-non-fempty-is-non-fempty*: ‹*A* ≠ {||} ⟹ *f* |‘| *A* ≠ {||}›
  unfolding *fimage-is-fempty*
  by *blast*

lemma *Union-empty-if-set-empty-or-all-empty*:

‹*ffUnion A* = {||} ⟹ *A* = {||} ∨ *fBall A* (λ *x*. *x* = {||})›
by (*metis* (*mono-tags*, *lifting*) *ffunion-insert finsert-absorb funion-fempty-right*)

lemma *fBall-fimage-is-fBall*: ‹*fBall* (*f* |‘| *A*) *P* ⟷ *fBall A* (λ *x*. *P* (*f x*))›
by *auto*

lemma *fset-map2*: ‹*v* ∈ *fset A* ⟹ *g* (*f v*) ∈ *set* (*map g* (*map f* (*list-of-fset A*)))›
proof −
assume ‹*v* ∈ *fset A*›
then show ‹*g* (*f v*) ∈ *set* (*map g* (*map f* (*list-of-fset A*)))›
unfolding *list-of-fset-def*
by (*smt* (*verit*, *ccfv-SIG*) *exists-fset-of-list fset-of-list.rep-eq imageI list.set-map someI-ex*)
qed

end

theory *Disjunctive-Consequence-Relations*
imports *Saturation-Framework.Calculus*
Propositional-Proof-Systems.Compactness
HOL−Library.Library
HOL−Library.Product-Lexorder
Lazy-List-Limsup
FSet-Extra
begin

2 Disjunctive Consequence Relations

no-notation *Sema.formula-semantics* (**infix** ⊨ *51*)

locale *consequence-relation* =
fixes
bot :: *'f* **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨ *50*)
assumes
bot-entails-empty: {*bot*} ⊨ {} **and**
entails-reflexive: {*C*} ⊨ {*C*} **and**
entails-subsets: *M'* ⊆ *M* ⟹ *N'* ⊆ *N* ⟹ *M'* ⊨ *N'* ⟹ *M* ⊨ *N* **and**
entails-cut: *M* ⊨ *N* ∪ {*C*} ⟹ *M'* ∪ {*C*} ⊨ *N'* ⟹ *M* ∪ *M'* ⊨ *N* ∪ *N'* **and**
entails-compactness: *M* ⊨ *N* ⟹ ∃ *M' N'*. (*M'* ⊆ *M* ∧ *N'* ⊆ *N* ∧ *finite M'* ∧ *finite N'* ∧ *M'* ⊨ *N'*)

begin

definition *order-double-subsets* :: ('f set * 'f set) ⇒ ('f set * 'f set) ⇒ bool
(**infix** $\preceq_s$ 50) **where**
‹($\preceq_s$) ≡ λC1 C2. fst C1 ⊆ fst C2 ∧ snd C1 ⊆ snd C2›
definition *order-double-subsets-strict* :: ('f set * 'f set) ⇒ ('f set * 'f set) ⇒ bool
(**infix** $\prec_s$ 50) **where**
‹($\prec_s$) ≡ λC1 C2. C1 $\preceq_s$ C2 ∧ C1 ≠ C2›

lemma *trivial-induction-order* : ‹C1 ⊆ B ∧ C2 ⊆ B' ⟶ (C1,C2) $\preceq_s$ (B,B')›
unfolding *order-double-subsets-def*
by *simp*

lemma *zorn-relation-trans* : ‹∀ C1 C2 C3. (C1 $\preceq_s$ C2) ⟶ (C2 $\preceq_s$ C3) ⟶ (C1 $\preceq_s$ C3)›
proof −
have ‹∀ C1 C2 C3. fst C1 ⊆ fst C2 ⟶ fst C2 ⊆ fst C3 ⟶ fst C1 ⊆ fst C3›
by *blast*
then have ‹∀ C1 C2 C3. snd C1 ⊆ snd C2 ⟶ snd C2 ⊆ snd C3 ⟶ snd C1 ⊆ snd C3›
by *blast*
then show *?thesis*
unfolding *order-double-subsets-def*
by *auto*
qed

lemma *zorn-strict-relation-trans* :
‹∀(C1::'f set × 'f set) C2 C3. (C1 $\prec_s$ C2) ⟶ (C2 $\prec_s$ C3) ⟶ (C1 $\prec_s$ C3)›
by (*metis order-double-subsets-def order-double-subsets-strict-def prod.expand subset-antisym*
zorn-relation-trans)

lemma *zorn-relation-antisym* : ‹∀ C1 C2. (C1 $\preceq_s$ C2) ⟶ (C2 $\preceq_s$ C1) ⟶ (C1 = C2)›
proof −
have ‹∀ C1 C2. (fst C1 ⊆ fst C2) ⟶ (fst C2 ⊆ fst C1) ⟶ (fst C1 = fst C2)›
by *force*
then have ‹∀ C1 C2. (snd C1 ⊆ snd C2) ⟶ (snd C2 ⊆ snd C1) ⟶ (snd C1 = snd C2)›
by *force*
then show *?thesis*
unfolding *order-double-subsets-def*
using *dual-order.eq-iff*
by *auto*
qed

lemma *entails-supsets* : (∀ M' N'. (M' ⊇ M ∧ N' ⊇ N ∧ M' ∪ N' = UNIV) ⟶ M' ⊨ N') ⟹ M ⊨ N

proof (*rule ccontr*)
assume
not-M-entails-N : ‹$\neg M \models N$› **and**
hyp-entails-sup : ‹$(\forall M'\ N'.\ (M' \supseteq M \wedge N' \supseteq N \wedge M' \cup N' = UNIV) \longrightarrow M' \models N')$›
have *contrapos-hyp-entails-sup*: ‹$\exists M'\ N'.\ (M' \supseteq M \wedge N' \supseteq N \wedge M' \cup N' = UNIV) \wedge \neg M' \models N'$›
proof −
define A :: $('f\ set * 'f\ set)\ set$ **where** ‹$A = \{(M',N').\ M \subseteq M' \wedge N \subseteq N' \wedge \neg M' \models N'\}$›
define *zorn-relation* :: $(('f\ set * 'f\ set) \times ('f\ set * 'f\ set))\ set$ **where**
‹*zorn-relation* = $\{(C1,C2) \in A \times A.\ C1 \preceq_s C2\}$›
define *max-chain* :: $('f\ set * 'f\ set)\ set \Rightarrow 'f\ set * 'f\ set$ **where**
‹*max-chain* = $(\lambda C.$ *if* $C = \{\}$ *then* (M,N)
else $(\bigcup\{C1.\ \exists C2.\ (C1,C2) \in C\}, \bigcup\{C2.\ \exists C1.\ (C1,C2) \in C\}))$›

have *relation-in-A* : ‹$\bigwedge C.\ C \in Chains\ zorn\text{-}relation \Longrightarrow \forall\ C1 \in C.\ C1 \in A$›
using *in-ChainsD zorn-relation-def*
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq mem-Sigma-iff old.prod.case*)
have *M-N-in-A* : ‹$(M,N) \in A$›
using *not-M-entails-N A-def* **by** *simp*
then have *not-empty-A* : ‹$A \neq \{\}$›
by *force*

have *trivial-replacement-order* [*simp*] : ‹$\forall\ C1\ C2.\ (C1,C2) \in zorn\text{-}relation \longrightarrow (C1 \preceq_s C2)$›
unfolding *zorn-relation-def* **by** *force*
moreover have *zorn-relation-refl* : ‹$\forall\ C \in A.\ C \preceq_s C$›
proof −
have ‹$\forall\ C \in A.\ fst\ C \subseteq fst\ C \wedge snd\ C \subseteq snd\ C$›
by *blast*
then show *?thesis*
unfolding *order-double-subsets-def*
by *simp*
qed
moreover have *refl-on-zorn-relation* : *refl-on A zorn-relation*
using *zorn-relation-refl*
by (*smt* (*verit*, *ccfv-SIG*) *case-prod-conv mem-Collect-eq mem-Sigma-iff refl-onI*
subrelI zorn-relation-def)
moreover have *zorn-relation-field-is-A* : *Field zorn-relation* = *A*
proof −
have ‹$\forall\ C0 \in A.\ (M,N) \preceq_s C0$›
unfolding *order-double-subsets-def*
using *A-def* **by** *simp*
then have ‹$\forall\ C0 \in A.\ ((M,N),C0) \in zorn\text{-}relation$›

```
      unfolding zorn-relation-def
      using M-N-in-A by simp
    then have A ⊆ Range zorn-relation
      unfolding Range-def by fast
    moreover have ‹∀ C0. C0 ∈ (Range zorn-relation) ⟶ C0 ∈ A›
      unfolding Range-iff using refl-on-zorn-relation zorn-relation-def by blast
    moreover have ‹∀ C0. C0 ∈ (Domain zorn-relation) ⟶ C0 ∈ A›
      using zorn-relation-def by blast
    ultimately show ?thesis
      by (metis Field-def Un-absorb1 subrelI subset-antisym)
  qed
  ultimately have zorn-hypothesis-po: Partial-order zorn-relation
  proof −
    have antisym-zorn-relation : antisym zorn-relation
    proof −
      have ‹∀ C1 C2. (C1,C2) ∈ zorn-relation ∧ (C2,C1) ∈ zorn-relation ⟶
(C1 ≼s C2) ∧ (C2 ≼s C1)›
        by force
      then show ?thesis using zorn-relation-antisym
        by (meson antisymI)
    qed

    moreover have trans zorn-relation
    proof −
      have ‹∀ C1 C2 C3. (C1,C2) ∈ zorn-relation
            ⟶ (C2,C3) ∈ zorn-relation ⟶ (C1 ≼s C2) ⟶ (C2 ≼s C3)›
        unfolding zorn-relation-def by blast
      then  have ‹∀ C1∈A. ∀ C2∈A. (C1 ≼s C2) ⟶ (C1,C2) ∈ zorn-relation›
        unfolding zorn-relation-def by blast
      then show ?thesis using zorn-relation-trans
        by (metis (no-types, opaque-lifting) FieldI1 FieldI2 transI
          trivial-replacement-order zorn-relation-field-is-A zorn-relation-trans)
    qed

    ultimately have preorder-on A zorn-relation
      unfolding preorder-on-def refl-on-zorn-relation
      using refl-on-zorn-relation
      using zorn-relation-def by blast
    then have partial-order-on A zorn-relation
      unfolding partial-order-on-def
      using antisym-zorn-relation by simp
    moreover have zorn-relation-field :  Field zorn-relation = A
    proof −
      have ‹∀  C0 ∈ A. (M,N) ≼s C0›
        unfolding order-double-subsets-def
        using A-def by simp
      then have ‹∀  C0 ∈ A. ((M,N),C0) ∈ zorn-relation›
        unfolding zorn-relation-def
        using M-N-in-A by simp
```

then have $A \subseteq$ *Range zorn-relation*
unfolding *Range-def* **by** *fast*
moreover have ‹$\forall$ *C0*. *C0* $\in$ (*Range zorn-relation*) $\longrightarrow$ *C0* $\in$ *A*›
unfolding *Range-iff* **using** *refl-on-zorn-relation zorn-relation-def* **by** *blast*
moreover have ‹$\forall$ *C0*. *C0* $\in$ (*Domain zorn-relation*) $\longrightarrow$ *C0* $\in$ *A*›
using *zorn-relation-def* **by** *blast*
ultimately show *?thesis*
by (*metis Field-def Un-absorb1 subrelI subset-antisym*)
qed

show *?thesis* **using** *zorn-relation-field calculation* **by** *simp*
qed

have *max-chain-is-a-max* : ‹$\bigwedge$*C*. *C* $\in$ *Chains zorn-relation* $\Longrightarrow$ $\forall$ *C1*$\in$*C*. (*C1* $\preceq_s$ *max-chain C*)›
proof −
fix *C*
assume *C-is-a-chain* : ‹*C* $\in$ *Chains zorn-relation*›
consider (*a*) *C* = {} | (*b*) *C* $\neq$ {}
by *auto*
then show ‹$\forall$ *C1* $\in$ *C*. *C1* $\preceq_s$ *max-chain C*›
proof *cases*
case *a*
show *?thesis* **by** (*simp add*: *a*)
next
case *b*
have ‹*C* $\subseteq$ *A*›
using *C-is-a-chain relation-in-A* **by** *blast*
then have ‹$\forall$ *C1* $\in$ *C*. $\exists$ (*C2*,*C3*) $\in$ *C*. *C1* = (*C2*,*C3*)›
by *blast*
moreover have ‹$\forall$ (*C1*,*C2*) $\in$ *C*. *C1* $\subseteq \bigcup\{C3.\ \exists\ C4.\ (C3,C4) \in C\}$›
by *blast*
moreover have ‹$\forall$ (*C1*,*C2*) $\in$ *C*. *C2* $\subseteq \bigcup\{C4.\ \exists\ C3.\ (C3,C4) \in C\}$›
by *blast*
moreover have ‹$\forall$ (*C1*,*C2*) $\in$ *C*.
((*C1* $\subseteq \bigcup\{C3.\ \exists\ C4.\ (C3,C4) \in C\} \wedge$ *C2* $\subseteq \bigcup\{C4.\ \exists\ C3.\ (C3,C4) \in C\}) \longrightarrow$
(*C1*,*C2*) $\preceq_s$ ($\bigcup\{C3.\ \exists\ C4.\ (C3,C4) \in C\}$, $\bigcup\{C4.\ \exists\ C3.\ (C3,C4) \in C\}$)›
unfolding *order-double-subsets-def*
using *trivial-induction-order*
by *simp*
ultimately have ‹$\forall$ (*C1*,*C2*) $\in$ *C*.
(*C1*,*C2*) $\preceq_s$ ($\bigcup\{C3.\ \exists\ C4.\ (C3,C4) \in C\}$,$\bigcup\{C4.\ \exists\ C3.\ (C3,C4) \in C\}$)›
by *fastforce*
then have ‹$\forall$ (*C1*,*C2*) $\in$ *C*. (*C1*,*C2*) $\preceq_s$ *max-chain C*›
unfolding *max-chain-def*

```
      by simp
    then show ‹∀ C1 ∈ C. C1 ⪯s max-chain C›
      by fast
  qed
qed


  have M-N-less-than-max-chain : ‹⋀C. C ∈ Chains zorn-relation ⟹ (M,N)
⪯s max-chain C›
  proof −
    fix C
    assume C-chain : ‹C ∈ Chains zorn-relation›
    consider (a) C = {} | (b) C ≠ {}
      by blast
    then show ‹(M,N) ⪯s max-chain C›
    proof cases
      case a
      assume C = {}
      then have ‹max-chain C = (M,N)›
        unfolding max-chain-def
        by simp
      then show ‹(M,N) ⪯s max-chain C›
        using M-N-in-A zorn-relation-refl
        by simp
    next
      case b
      assume C-not-empty : C ≠ {}
      have M-minor-first : ‹∀ C1 ∈ C. M ⊆ fst C1›
        using A-def C-chain relation-in-A by fastforce
      have N-minor-second : ‹∀ C1 ∈ C. N ⊆ snd C1›
        using A-def C-chain relation-in-A by fastforce
       moreover have ‹(∀ C1 ∈ C. (fst C1 ⊆ fst (max-chain C)) ∧ snd C1 ⊆
snd (max-chain C))›
        using order-double-subsets-def C-chain max-chain-is-a-max
        by presburger
      moreover have ‹(∃ C1 ∈ C. (M ⊆ fst C1 ∧ N ⊆ snd C1)
                    ⟶ fst C1 ⊆ fst (max-chain C) ∧ snd C1 ⊆ snd (max-chain
C))
                  ⟶ M ⊆ fst (max-chain C) ∧ N ⊆ snd (max-chain C)›
        using M-minor-first N-minor-second
        by blast
      ultimately have ‹M ⊆ fst (max-chain C) ∧ N ⊆ snd (max-chain C)›
          by (meson order-double-subsets-def C-chain C-not-empty ex-in-conv
max-chain-is-a-max)
      then show ‹(M,N) ⪯s max-chain C›
        unfolding order-double-subsets-def
        by simp
    qed
  qed
```

moreover have *left-U-not-entails-right-U*:
‹$\bigwedge C.\ C \in$ *Chains zorn-relation* $\Longrightarrow \neg$ *fst* (*max-chain* C)$\models$ *snd* (*max-chain* C)›
proof −
fix C
assume *C-chain* :‹$C \in$ *Chains zorn-relation*›
consider (*a*) $C = \{\}$ | (*b*) $C \neq \{\}$
by *fast*
then show ‹$\neg$ *fst* (*max-chain* C)$\models$ *snd* (*max-chain* C)›
proof *cases*
case *a*
then show *?thesis* **using** *not-M-entails-N*
unfolding *max-chain-def*
by *simp*
next
case *b*
assume *C-not-empty* :‹$C \neq \{\}$›
show *?thesis*
proof (*rule ccontr*)
assume ‹$\neg\neg$*fst* (*max-chain* C) $\models$ *snd* (*max-chain* C)›
then have *abs-fst-entails-snd* : ‹*fst* (*max-chain* C) $\models$ *snd* (*max-chain* C)›
by *auto*
then obtain M' N' **where**
abs-max-chain-compactness : ‹$M' \subseteq$ *fst* (*max-chain* C)
$\wedge$ $N' \subseteq$ *snd* (*max-chain* C)
$\wedge$ *finite* M'
$\wedge$ *finite* N'
$\wedge$ $M' \models N'$›
using *entails-compactness* **by** *fastforce*
then have *not-empty-M'-or-N'* : ‹($M' \neq \{\}$) $\vee$ ($N' \neq \{\}$)›
by (*meson empty-subsetI entails-subsets not-M-entails-N*)
then have *finite-M'-subset* : ‹(*finite* M') $\wedge$ $M' \subseteq \bigcup\{C1.\ \exists C2.\ (C1,C2) \in C\}$›
using *C-not-empty abs-max-chain-compactness max-chain-def*
by *simp*
then have *M'-in-great-union* : ‹$M' \subseteq \bigcup\{C1.\ \exists C2.\ (C1,C2) \in C \wedge C1 \cap M' \neq \{\}\}$›
by *blast*
then have *M'-in-finite-union* :
‹$\exists P \subseteq \{C1.\ \exists C2.\ (C1,C2) \in C \wedge C1 \cap M' \neq \{\}\}.$ *finite* $P \wedge M' \subseteq \bigcup P$›
by (*meson finite-M'-subset finite-subset-Union*)
moreover have *finite-N'-subset* : ‹(*finite* N') $\wedge$ $N' \subseteq \bigcup\{C2.\ \exists C1.\ (C1,C2) \in C\}$›
using *C-not-empty abs-max-chain-compactness*
using *max-chain-def*
by *simp*
then have *N'-in-great-union* : ‹$N' \subseteq \bigcup\{C2.\ \exists C1.\ (C1,C2) \in C \wedge C2 \cap N' \neq \{\}\}$›

by *blast*
then have *N′-in-finite-union* :
‹∃ *Q* ⊆ {*C2*. ∃ *C1*. (*C1*,*C2*) ∈ *C* ∧ *C2* ∩ *N′* ≠ {}}. *finite Q* ∧ *N′* ⊆ ⋃ *Q*›
by (*meson finite-N′-subset finite-subset-Union*)

ultimately obtain *P Q* **where**
P-subset : ‹*P* ⊆ {*C1*. ∃ *C2*. (*C1*,*C2*) ∈ *C* ∧ *C1* ∩ *M′* ≠ {}}› **and**
finite-P: ‹*finite P*› **and**
P-supset : ‹*M′* ⊆ ⋃ *P*› **and**
Q-subset : ‹*Q* ⊆ {*C2*. ∃ *C1*. (*C1*,*C2*) ∈ *C* ∧ *C2* ∩ *N′* ≠ {}}›
and *finite-Q* : ‹*finite Q*› **and** *Q-supset* : ‹*N′* ⊆ ⋃ *Q*›
by *auto*
have *not-empty-P-or-Q* : ‹*P* ≠ {} ∨ *Q* ≠ {}›
using *not-empty-M′-or-N′ P-supset Q-supset* **by** *blast*
have *P-linked-C* : ‹∀ *C1*∈*P*. ∃ *C2*. (*C1*,*C2*)∈*C*›
using *P-subset* **by** *auto*
then have *Q-linked-C* : ‹∀ *C2*∈*Q*. ∃ *C1*. (*C1*,*C2*)∈*C*›
using *Q-subset* **by** *auto*
then have ‹∃ *P* ⊆ *C*. *P* = {(*C1*,*C2*). (*C1*,*C2*) ∈ *C* ∧ *C1* ∈ *P*}›
by *fastforce*

then obtain *P* **where**
P-def: ‹*P* = {(*C1*,*C2*). (*C1*,*C2*) ∈ *C* ∧ *C1* ∈ *P*}›
by *auto*
then have *P-in-C* : ‹*P* ⊆ *C*›
by *auto*
have *P-linked-P* : ‹∀ *C1*∈*P*. ∃ *C2*. (*C1*,*C2*)∈*P*›
using *P-linked-C P-def*
by *simp*

define *f* **where**
‹*f* = (λ*C1*. *if* (∃ *C2*. (*C1*,*C2*)∈*P*) *then* (*SOME C2*. (*C1*,*C2*) ∈ *P*) *else* {})›
have ‹∀ (*C1*,*C2*)∈*P*. *C1*∈*P*›
using *P-def* **by** *blast*
have *f-stability-in-P* : ‹⋀ *C1 C2*. (*C1*,*C2*)∈*P* ⟹ (*C1*, *f C1*)∈*P*›
proof −
fix *C1 C2*
show ‹(*C1*,*C2*)∈*P* ⟹ (*C1*, *f C1*)∈*P*›
proof −
assume *C1-C2-in-P* : ‹(*C1*,*C2*)∈*P*›
then have ‹∃ *C3*. (*C1*,*C3*)∈*P*›
by *blast*
then have ‹(*C1*, *f C1*) = (*C1*, *SOME C3*. (*C1*,*C3*) ∈ *P*)›
unfolding *f-def* **by** *simp*
then have ‹(*C1*, *f C1*)∈*P*›
by (*metis C1-C2-in-P someI-ex*)
then show ‹(*C1*,*C2*)∈*P* ⟹ (*C1*, *f C1*)∈*P*› **by** *blast*

qed
qed

define $\mathcal{P}'$ **where**
$\langle \mathcal{P}' = \{(C1,\ f\ C1) | C1.\ \exists\ C2.\ (C1,C2) \in \mathcal{P}\} \rangle$
then have *injectivity-*$\mathcal{P}'$: $\langle \forall\ C1\ C2\ C2'.(((C1,\ C2) \in \mathcal{P}' \wedge (C1,\ C2') \in \mathcal{P}') \longrightarrow C2 = C2') \rangle$
by *auto*
have $\mathcal{P}'$*-in-*$\mathcal{P}$: $\langle \mathcal{P}' \subseteq \mathcal{P} \rangle$
unfolding $\mathcal{P}'$*-def*
using *f-stability-in-*$\mathcal{P}$
by *blast*
then have $\mathcal{P}'$*-in-C* : $\langle \mathcal{P}' \subseteq C \rangle$
using $\mathcal{P}$*-in-C*
by *blast*
have $\mathcal{P}'$*-less-than-P* : $\langle \forall\ C0 \in \mathcal{P}'.\ fst\ C0 \in P \rangle$
unfolding $\mathcal{P}'$*-def* $\mathcal{P}$*-def* **by** *fastforce*
have *P-less-than-*$\mathcal{P}$: $\langle \forall\ C1 \in P.\ \exists\ C2.\ (C1,C2) \in \mathcal{P} \rangle$
unfolding $\mathcal{P}$*-def*
using *P-linked-C* **by** *simp*
then have *P-less-than-*$\mathcal{P}$*-reformulated* : $\langle \forall\ C1 \in P.\ \exists\ C0 \in \mathcal{P}'.\ (fst\ C0) = C1 \rangle$
unfolding $\mathcal{P}'$*-def*
by *simp*
then have *union-P-in-union-fst-*$\mathcal{P}'$: $\langle \bigcup P \subseteq \bigcup \{C1.\ \exists\ C0 \in \mathcal{P}'.\ (fst\ C0) = C1\} \rangle$
using *Union-mono subsetI*
by *fastforce*
have *injectivity-*$\mathcal{P}'$*-reformulated* :
$\langle \forall\ C0\ C0'.\ ((C0 \in \mathcal{P}' \wedge C0' \in \mathcal{P}' \wedge C0' \neq C0) \longrightarrow (fst\ C0) \neq (fst\ C0')) \rangle$
unfolding $\mathcal{P}'$*-def*
by *force*

define *bij-*$\mathcal{P}'$:: $('f\ set \times 'f\ set) \Rightarrow 'f\ set$ **where**
$\langle bij\text{-}\mathcal{P}' \equiv fst \rangle$
have *injectivity-bij-*$\mathcal{P}'$: $\langle \forall\ C0 \in \mathcal{P}'.\ \forall\ C0' \in \mathcal{P}'.\ bij\text{-}\mathcal{P}'\ C0 = bij\text{-}\mathcal{P}'\ C0' \longrightarrow C0 = C0' \rangle$
unfolding *bij-*$\mathcal{P}'$*-def*
using *injectivity-*$\mathcal{P}'$*-reformulated* **by** *blast*
then have *bij-*$\mathcal{P}'$*-injectivity* : $\langle inj\text{-}on\ bij\text{-}\mathcal{P}'\ \mathcal{P}' \rangle$
unfolding *inj-on-def*
by *simp*
moreover have *surjectivity-bij-*$\mathcal{P}'$*-first-inc* : $\langle bij\text{-}\mathcal{P}'\ `\ \mathcal{P}' \subseteq P \rangle$
unfolding *bij-*$\mathcal{P}'$*-def*
using $\mathcal{P}'$*-less-than-P image-subset-iff* **by** *auto*
moreover have *surjectivity-bij-*$\mathcal{Q}'$*-second-inc* : $\langle P \subseteq bij\text{-}\mathcal{P}'\ `\ \mathcal{P}' \rangle$
unfolding *bij-*$\mathcal{P}'$*-def*
using *P-less-than-*$\mathcal{P}$*-reformulated* **by** *auto*

ultimately have *surjectivity-bij-$\mathcal{P}'$* : ‹*bij-$\mathcal{P}'$* ‘ $\mathcal{P}'$ = *P*›
by *blast*
then have *bij* : ‹*bij-betw bij-$\mathcal{P}'$ $\mathcal{P}'$ P*›
unfolding *bij-betw-def*
using *bij-$\mathcal{P}'$-injectivity* **by** *simp*
then have *finite-$\mathcal{P}'$* : ‹*finite* $\mathcal{P}'$›
using *bij-$\mathcal{P}'$-injectivity finite-P inj-on-finite surjectivity-bij-$\mathcal{P}'$-first-inc*
by *auto*
moreover have ‹∃ $\mathcal{Q}$ ⊆ *C*. $\mathcal{Q}$ = {(*C1*,*C2*). (*C1*,*C2*) ∈ *C* ∧ *C2* ∈ *Q*}›
by *fastforce*

then obtain $\mathcal{Q}$ **where**
$\mathcal{Q}$-def: ‹$\mathcal{Q}$ = {(*C1*,*C2*). (*C1*,*C2*) ∈ *C* ∧ *C2* ∈ *Q*}›
by *auto*
then have *$\mathcal{Q}$-in-C* : ‹$\mathcal{Q}$ ⊆ *C*›
by *auto*

define *g* **where**
‹*g* = (λ*C2*. *if* (∃ *C1*. (*C1*,*C2*)∈$\mathcal{Q}$) *then* (*SOME C1*. (*C1*,*C2*) ∈ $\mathcal{Q}$) *else*
{})›
have ‹∀ (*C1*,*C2*)∈$\mathcal{Q}$. *C2*∈*Q*›
using *$\mathcal{Q}$-def* **by** *blast*
have *g-stability-in-$\mathcal{Q}$* : ‹⋀ *C1 C2*. (*C1*,*C2*)∈$\mathcal{Q}$ ⟹ (*g C2*, *C2*)∈$\mathcal{Q}$›
proof −
fix *C1 C2*
show ‹(*C1*,*C2*)∈$\mathcal{Q}$ ⟹ (*g C2*, *C2*)∈$\mathcal{Q}$›
proof −
assume *C1-C2-in-$\mathcal{Q}$* : ‹(*C1*,*C2*)∈$\mathcal{Q}$›
then have ‹∃ *C3*. (*C3*,*C2*)∈$\mathcal{Q}$›
by *blast*
then have ‹(*g C2*, *C2*) = ((*SOME C1*. (*C1*,*C2*) ∈ $\mathcal{Q}$), *C2*)›
unfolding *g-def* **by** *simp*
then have ‹(*g C2*, *C2*)∈$\mathcal{Q}$›
by (*metis C1-C2-in-$\mathcal{Q}$ someI-ex*)
then show ‹(*C1*,*C2*)∈$\mathcal{Q}$ ⟹ (*g C2*, *C2*)∈$\mathcal{Q}$› **by** *blast*
qed
qed

define $\mathcal{Q}'$ **where**
‹$\mathcal{Q}'$ = {(*g C2*, *C2*)|*C2*. ∃ *C1*. (*C1*,*C2*)∈$\mathcal{Q}$}›
then have *injectivity-$\mathcal{Q}'$* : ‹∀ *C1 C2 C1'*.(((*C1*, *C2*) ∈ $\mathcal{Q}'$ ∧ (*C1'*, *C2*) ∈
$\mathcal{Q}'$) ⟶ *C1* = *C1'*)›
by *auto*
have *$\mathcal{Q}'$-in-$\mathcal{Q}$* : ‹$\mathcal{Q}'$ ⊆ $\mathcal{Q}$›
unfolding *$\mathcal{Q}'$-def*
using *g-stability-in-$\mathcal{Q}$*
by *blast*
then have *$\mathcal{Q}'$-in-C* : ‹$\mathcal{Q}'$ ⊆ *C*›
using *$\mathcal{Q}$-in-C*

by *blast*
have $\mathcal{Q}'$*-less-than-Q* : ‹$\forall\, C0 \in \mathcal{Q}'.\ snd\ C0 \in Q$›
unfolding $\mathcal{Q}'$*-def* $\mathcal{Q}$*-def* **by** *fastforce*
have *Q-less-than-*$\mathcal{Q}$: ‹$\forall\, C2 \in Q.\ \exists\, C1.\ (C1, C2) \in \mathcal{Q}$›
unfolding $\mathcal{Q}$*-def*
using *Q-linked-C* **by** *simp*
then have *Q-less-than-*$\mathcal{Q}$*-reformulated* :‹$\forall\, C2 \in Q.\ \exists\, C0 \in \mathcal{Q}'.\ (snd\ C0) = C2$›
unfolding $\mathcal{Q}'$*-def*
by *simp*
then have *union-Q-in-union-fst-*$\mathcal{Q}'$: ‹$\bigcup Q \subseteq \bigcup \{C2.\ \exists\, C0 \in \mathcal{Q}'.\ (snd\ C0) = C2\}$›
using *Union-mono subsetI*
by *fastforce*
have *injectivity-*$\mathcal{Q}'$*-reformulated* :
‹$\forall\, C0\ C0'.\ ((C0 \in \mathcal{Q}' \wedge C0' \in \mathcal{Q}' \wedge C0' \neq C0) \longrightarrow (snd\ C0) \neq (snd\ C0'))$›
unfolding $\mathcal{Q}'$*-def*
by *force*

define *bij-*$\mathcal{Q}'$:: $('f\ set \times 'f\ set) \Rightarrow 'f\ set$ **where**
‹*bij-*$\mathcal{Q}' \equiv snd$›
have *injectivity-bij-*$\mathcal{Q}'$: ‹$\forall\, C0 \in \mathcal{Q}'.\ \forall\, C0' \in \mathcal{Q}'.\ bij\text{-}\mathcal{Q}'\ C0 = bij\text{-}\mathcal{Q}'\ C0' \longrightarrow C0 = C0'$›
unfolding *bij-*$\mathcal{Q}'$*-def*
using *injectivity-*$\mathcal{Q}'$*-reformulated* **by** *blast*
then have *bij-*$\mathcal{Q}'$*-injectivity* : ‹*inj-on bij-*$\mathcal{Q}'$ $\mathcal{Q}'$›
unfolding *inj-on-def*
by *simp*
moreover have *surjectivity-bij-*$\mathcal{Q}'$*-first-inc* : ‹*bij-*$\mathcal{Q}'$ ‘ $\mathcal{Q}' \subseteq Q$›
unfolding *bij-*$\mathcal{Q}'$*-def*
using $\mathcal{Q}'$*-less-than-Q image-subset-iff* **by** *auto*
moreover have *surjectivity-bij-*$\mathcal{Q}'$*-second-inc* : ‹$Q \subseteq$ *bij-*$\mathcal{Q}'$ ‘ $\mathcal{Q}'$›
unfolding *bij-*$\mathcal{Q}'$*-def*
using *Q-less-than-*$\mathcal{Q}$*-reformulated* **by** *auto*
ultimately have *surjectivity-bij-*$\mathcal{Q}'$: ‹*bij-*$\mathcal{Q}'$ ‘ $\mathcal{Q}' = Q$›
by *blast*
then have *bij* : ‹*bij-betw bij-*$\mathcal{Q}'$ $\mathcal{Q}'$ Q›
unfolding *bij-betw-def*
using *bij-*$\mathcal{Q}'$*-injectivity* **by** *simp*
then have *finite-*$\mathcal{Q}'$: ‹*finite* $\mathcal{Q}'$›
using *bij-*$\mathcal{Q}'$*-injectivity finite-Q inj-on-finite surjectivity-bij-*$\mathcal{Q}'$*-first-inc* **by** *auto*

have *not-empty-*$\mathcal{P}$*-or-*$\mathcal{Q}$: ‹$\mathcal{P} \neq \{\} \vee \mathcal{Q} \neq \{\}$›
using $\mathcal{P}'$*-in-*$\mathcal{P}$ $\mathcal{Q}'$*-in-*$\mathcal{Q}$ *surjectivity-bij-*$\mathcal{P}'$ *surjectivity-bij-*$\mathcal{Q}'$ *not-empty-P-or-Q*

by *fastforce*
then have *not-empty-*$\mathcal{P}'$*-or-*$\mathcal{Q}'$: ‹$\mathcal{P}' \neq \{\} \vee \mathcal{Q}' \neq \{\}$›

using *not-empty-P-or-Q surjectivity-bij-$\mathcal{P}'$ surjectivity-bij-$\mathcal{Q}'$* **by** *blast*

define $\mathcal{R}'$ **where** ‹$\mathcal{R}' = \mathcal{P}' \cup \mathcal{Q}'$›
have ‹$\forall\ (C1,C2) \in \mathcal{R}'.\ C1 \in P \vee C2 \in Q$›
unfolding $\mathcal{R}'$*-def* $\mathcal{P}'$*-def* $\mathcal{Q}'$*-def* $\mathcal{P}$*-def* $\mathcal{Q}$*-def*
by *fastforce*
have *finite-*$\mathcal{R}'$: ‹*finite* $\mathcal{R}'$›
unfolding $\mathcal{R}'$*-def*
using *finite-*$\mathcal{P}'$ *finite-*$\mathcal{Q}'$ **by** *simp*
moreover have $\mathcal{R}'$*-in-C* : ‹$\mathcal{R}' \subseteq C$›
unfolding $\mathcal{R}'$*-def*
using $\mathcal{P}'$*-in-C* $\mathcal{Q}'$*-in-C*
by *blast*
moreover have *not-empty-*$\mathcal{R}'$: ‹$\mathcal{R}' \neq \{\}$›
using $\mathcal{R}'$*-def not-empty-*$\mathcal{P}'$*-or-*$\mathcal{Q}'$ **by** *blast*

have *max-*$\mathcal{R}'$: ‹$\exists (M0,N0) \in \mathcal{R}'.\ (\forall (M,N) \in \mathcal{R}'.\ (M,N) \preceq_s (M0,N0))$›
proof (*rule ccontr*)
assume ‹$\neg(\exists (M0,N0) \in \mathcal{R}'.\ (\forall (M,N) \in \mathcal{R}'.\ (M,N) \preceq_s (M0,N0)))$›
then have *abs-max-*$\mathcal{R}'$: ‹$\forall (M0,N0) \in \mathcal{R}'.\ (\exists (M,N) \in \mathcal{R}'.\ \neg((M,N) \preceq_s (M0,N0)))$›
by *auto*
have ‹$\forall (M0,N0) \in C.\ \forall (M,N) \in C.\ \neg((M,N),(M0,N0)) \in$ *zorn-relation* $\longrightarrow ((M0,N0),(M,N)) \in$ *zorn-relation*›
unfolding *zorn-relation-def*
using *C-chain*
by (*smt* (*verit*, *best*) *Chains-def case-prodI2 mem-Collect-eq zorn-relation-def*)
then have ‹$\forall (M0,N0) \in C.\ \forall (M,N) \in C.\ \neg(M,N) \preceq_s (M0,N0) \longrightarrow (M0,N0) \preceq_s (M,N)$›
unfolding *zorn-relation-def*
using *trivial-replacement-order*
by *blast*
then have ‹$\forall (M0,N0) \in \mathcal{R}'.\ \forall (M,N) \in \mathcal{R}'.\ \neg(M,N) \preceq_s (M0,N0) \longrightarrow (M0,N0) \preceq_s (M,N)$›
using $\mathcal{R}'$*-in-C*
by *auto*
then have ‹$\forall (M0,N0) \in \mathcal{R}'.\ \forall (M,N) \in \mathcal{R}'.\ \neg(M,N) \preceq_s (M0,N0) \longrightarrow (M0,N0) \prec_s (M,N)$›
unfolding *order-double-subsets-strict-def*
by *blast*
then have *abs-max-*$\mathcal{R}'$*-reformulated* : ‹$\forall (M0,N0) \in \mathcal{R}'.\ (\exists (M,N) \in \mathcal{R}'.\ (M0,N0) \prec_s (M,N))$›
using *abs-max-*$\mathcal{R}'$
by *blast*

define *find-dif* :: (*'f set* × *'f set*) ⇒ (*'f set* × *'f set*) **where**
‹*find-dif* = ($\lambda(M0,N0)$. *if* $(\exists (M,N) \in \mathcal{R}'.\ (M0,N0) \prec_s (M,N))$
then $(SOME\ (M,N).\ (M,N) \in \mathcal{R}' \wedge (M0,N0) \prec_s (M,N))$

```
                                    else ({},{}))›
          obtain M0 N0 where M0-N0-def : ‹(M0,N0)∈ ℛ'›
            using not-empty-ℛ' by auto
          define bij-nat :: nat ⇒ ('f set × 'f set) where
            ‹bij-nat ≡ λk. (find-dif ^^ k) (M0, N0)›
          have bij-nat-in-ℛ' : ‹bij-nat k ∈ ℛ'› for k
          proof (induction k)
            case 0
            then show ?case
              unfolding bij-nat-def
              using M0-N0-def
              by simp
          next
            case (Suc k)
            assume ‹bij-nat k ∈ ℛ'›
            have new-major-k : ‹∃(M,N)∈ℛ'. bij-nat k ≺s (M,N)›
              using abs-max-ℛ'-reformulated Suc
              by simp
           then have ‹find-dif (bij-nat k) = (SOME (M,N). (M,N)∈ℛ' ∧ bij-nat
k ≺s (M,N))›
              unfolding find-dif-def
              by auto
            then have ‹bij-nat (Suc k) = (SOME (M,N). (M,N)∈ℛ' ∧ bij-nat k
≺s (M,N))›
              unfolding bij-nat-def
              by simp
            then show ?case
             by (metis (mono-tags, lifting) new-major-k case-prod-conv some-eq-imp
surj-pair)
          qed
         then have new-major-general : ‹∃(M,N)∈ℛ'. (bij-nat k) ≺s (M,N)› for
k
            using abs-max-ℛ'-reformulated
            by simp
          then have ‹find-dif (bij-nat k) = (SOME (M,N). (M,N)∈ℛ' ∧ bij-nat
k ≺s (M,N))› for k
            unfolding find-dif-def
            by auto
          then have ‹bij-nat k ≺s find-dif (bij-nat k)› for k
                by (metis (mono-tags, lifting) case-prod-conv new-major-general
some-eq-imp surj-pair)
          then have bij-nat-croiss: ‹(bij-nat (k::nat)) ≺s (bij-nat (Suc k))› for k
            using bij-nat-def by simp
          have bij-nat-general-croiss : ‹i < j ⟹ bij-nat i ≺s bij-nat j› for i j
          proof −
            assume hyp-croiss : ‹i < j›
            have ‹bij-nat i ≺s bij-nat (i+1+(k::nat))› for k
            proof (induction k)
              case 0
```

```
    then show ?case using bij-nat-croiss by simp
  next
    case (Suc k)
    have ‹bij-nat (i+1+k) ≺s bij-nat (i+1+(Suc k))›
      using bij-nat-croiss by simp
    then show ?case using zorn-strict-relation-trans Suc by blast
  qed

  then have ‹bij-nat i ≺s bij-nat j›
 by (metis Suc-eq-plus1-left hyp-croiss add.assoc add.commute less-natE)
  then show ?thesis by simp
qed

have ‹bij-nat i = bij-nat j ∧ ¬i=j ⟹ False› for i j
proof −
  assume bij-nat-i-equals-bij-nat-j : ‹bij-nat i = bij-nat j ∧ ¬i=j›
  then have ‹i < j ∨ j < i›
    by auto
  then have ‹bij-nat i ≺s bij-nat j ∨ bij-nat j ≺s bij-nat i›
    using bij-nat-general-croiss
    by blast
  then have ‹bij-nat i ≠ bij-nat j›
    unfolding order-double-subsets-strict-def
    by force
  then show ?thesis
    using bij-nat-i-equals-bij-nat-j by simp
qed

then have bij-nat-inj : ‹bij-nat i = bij-nat j ⟶ i = j› for i j
  unfolding bij-nat-def
  by auto
then have bij-nat-inj-gen : ‹∀ i j. bij-nat i = bij-nat j ⟶ i = j›
  by auto
then have ‹inj-on bij-nat (UNIV :: nat set)›
  unfolding inj-on-def
  by simp
then have bij-nat-is-a-bij :
        ‹bij-betw bij-nat (UNIV :: nat set) (bij-nat ‘(UNIV :: nat set))›
  unfolding bij-betw-def
  by simp
 then have ‹finite (UNIV :: nat set) ⟷ finite (bij-nat ‘(UNIV:: nat
set))›
  using bij-betw-finite by auto
moreover have ‹¬(finite (UNIV :: nat set))›
  by simp
ultimately have ‹¬finite (bij-nat ‘(UNIV:: nat set))›
  by blast
moreover have ‹bij-nat ‘(UNIV:: nat set) ⊆ ℛ'›
  unfolding bij-nat-def
```

using $\mathcal{R}'$*-in-C bij-nat-in-*$\mathcal{R}'$ *image-subset-iff bij-nat-def*
by *blast*
ultimately have ‹¬(*finite* $\mathcal{R}'$)›
using *finite-subset* **by** *blast*
then show *False* **using** *finite-*$\mathcal{R}'$ **by** *blast*
qed

then obtain *M-max N-max* **where**
M-N-max-def : ‹(*M-max*,*N-max*)∈$\mathcal{R}'$ ∧ (∀ (*M*,*N*)∈$\mathcal{R}'$. (*M*,*N*) $\preceq_s$ (*M-max*,*N-max*))›
by *auto*
then have ‹∀ *C1*∈*P*. ∃ (*M0*,*N0*)∈$\mathcal{R}'$. *M0* = *C1* ›
using $\mathcal{R}'$*-def P-less-than-*$\mathcal{P}$*-reformulated* **by** *fastforce*
then have ‹$\bigcup P \subseteq \bigcup \{C1.\ \exists\, C0 \in \mathcal{R}'.\ (fst\ C0) = C1\}$›
unfolding $\mathcal{R}'$*-def*
by *fastforce*
moreover have ‹∀ *C1*. (∃ *C0* ∈ $\mathcal{R}'$. (*fst C0*) = *C1* ∧ *C0* $\preceq_s$ (*M-max*,*N-max*)) ⟶ *C1* ⊆ *M-max*›
unfolding *M-N-max-def order-double-subsets-def*
by *auto*
then have ‹∀ *C1* ∈ {*C1*. ∃ *C0* ∈ $\mathcal{R}'$. (*fst C0*) = *C1*}. *C1* ⊆ *M-max* ›
using *M-N-max-def* **by** *auto*
then have ‹$\bigcup \{C1.\ \exists\, C0 \in \mathcal{R}'.\ (fst\ C0) = C1\} \subseteq$ *M-max*›
by *auto*
ultimately have *union-P-in-M-max* : ‹$\bigcup P \subseteq$ *M-max*›
by *blast*
moreover have ‹∀ *C2*∈*Q*. ∃ (*M0*,*N0*)∈$\mathcal{R}'$. *N0* = *C2* ›
using $\mathcal{R}'$*-def Q-less-than-*$\mathcal{Q}$*-reformulated* **by** *fastforce*
then have ‹$\bigcup Q \subseteq \bigcup \{C2.\ \exists\, C0 \in \mathcal{R}'.\ (snd\ C0) = C2\}$›
unfolding $\mathcal{R}'$*-def*
by *fastforce*
moreover have ‹∀ *C2*. (∃ *C0* ∈ $\mathcal{R}'$. (*snd C0*) = *C2* ∧ *C0* $\preceq_s$ (*M-max*,*N-max*)) ⟶ *C2* ⊆ *N-max*›
unfolding *M-N-max-def order-double-subsets-def*
by *auto*
then have ‹∀ *C2* ∈ {*C2*. ∃ *C0* ∈ $\mathcal{R}'$. (*snd C0*) = *C2*}. *C2* ⊆ *N-max* ›
using *M-N-max-def* **by** *auto*
then have ‹$\bigcup \{C2.\ \exists\, C0 \in \mathcal{R}'.\ (snd\ C0) = C2\} \subseteq$ *N-max*›
by *auto*
ultimately have *union-Q-in-N-max* : ‹$\bigcup Q \subseteq$ *N-max*›
by *blast*
have ‹*M'* ⊆ *M-max* ∧ *N'* ⊆ *N-max*›
using *P-supset Q-supset union-P-in-M-max union-Q-in-N-max* **by** *auto*
then have ‹*M-max* ⊨ *N-max*›
using *abs-max-chain-compactness entails-subsets*
by *force*
moreover have ‹(*M-max*,*N-max*) ∈ $\mathcal{R}'$›
using *M-N-max-def*
by *simp*

then have ‹(*M-max*,*N-max*) ∈ *C*›
using $\mathcal{R}'$*-in-C* **by** *auto*
then have ‹(*M-max*,*N-max*) ∈ *A*›
using *C-chain relation-in-A* **by** *auto*
then have ‹¬*M-max* ⊨ *N-max*›
unfolding *A-def*
by *auto*
ultimately show *False*
by *simp*
qed
qed
qed
moreover have ‹⋀*C*. *C* ∈ *Chains zorn-relation* ⟹ *M* ⊆ *fst* (*max-chain C*)›
using *M-N-less-than-max-chain order-double-subsets-def fst-eqD*
by *metis*
moreover have ‹⋀*C*. *C* ∈ *Chains zorn-relation* ⟹ *N* ⊆ *snd* (*max-chain C*)›
using *M-N-less-than-max-chain order-double-subsets-def snd-eqD*
by *metis*
ultimately have *max-chain-in-A* : ‹⋀*C*. *C* ∈ *Chains zorn-relation* ⟹ *max-chain C* ∈ *A*›
unfolding *A-def* **using** *M-N-less-than-max-chain case-prod-beta*
by *force*

then have ‹⋀*C*. *C* ∈ *Chains zorn-relation* ⟹
(*max-chain C*) ∈ *A* ∧ (∀ *C0*∈*C*. (*C0*, *max-chain C*) ∈ *zorn-relation*)›
unfolding *zorn-relation-def*
using *zorn-relation-field-is-A max-chain-is-a-max relation-in-A zorn-relation-def*
by *fastforce*
then have *zorn-hypothesis-u* :
‹⋀*C*. *C* ∈ *Chains zorn-relation* ⟹ ∃ *u*∈*Field zorn-relation*. ∀ *a*∈*C*. (*a*, *u*) ∈ *zorn-relation*›
using *zorn-relation-field-is-A max-chain-is-a-max* **by** *auto*

then have ‹∃ *Cmax*∈*Field zorn-relation*. ∀ *C*∈*Field zorn-relation*.
(*Cmax*, *C*) ∈ *zorn-relation* ⟶ *C* = *Cmax*›
using *Zorns-po-lemma zorn-hypothesis-u zorn-hypothesis-po* **by** *blast*
then have *zorn-result* : ‹∃ *Cmax*∈*A*. ∀ *C*∈*A*. (*Cmax*, *C*) ∈ *zorn-relation* ⟶ *C* = *Cmax*›
using *zorn-relation-field-is-A* **by** *blast*
then obtain *Cmax* **where**
Cmax-in-A : ‹*Cmax*∈*A*› **and**
Cmax-is-max : ‹∀ *C*∈*A*. (*Cmax*, *C*) ∈ *zorn-relation* ⟶ *C* = *Cmax*›
by *blast*

have *Cmax-not-entails* : ‹¬ *fst Cmax* ⊨ *snd Cmax*›
unfolding *A-def*

```
    using Cmax-in-A
    using A-def by force
  have M-less-fst-Cmax : ‹M ⊆ fst Cmax›
    using A-def Cmax-in-A  by force
  moreover have N-less-snd-Cmax : ‹N ⊆ snd Cmax›
    using A-def Cmax-in-A  by force
  have ‹fst Cmax ∪ snd Cmax = UNIV›
  proof (rule ccontr)
    assume ‹¬fst Cmax ∪ snd Cmax = UNIV›
    then have ‹∃ C0. C0 ∉ fst Cmax ∪ snd Cmax›
      by auto
    then obtain C0 where C0-def : ‹C0 ∉ fst Cmax ∪ snd Cmax›
      by auto
    have fst-max-entailment-extended : ‹(fst Cmax) ∪ {C0} ⊨ snd Cmax›
    proof (rule ccontr)
      assume ‹¬(fst Cmax) ∪ {C0} ⊨ snd Cmax›
     then have fst-extended-Cmax-in-A : ‹((fst Cmax ∪ {C0}), snd Cmax) ∈ A›
        unfolding A-def
        using M-less-fst-Cmax N-less-snd-Cmax by blast
      then have ‹(Cmax,((fst Cmax) ∪ {C0},snd Cmax)) ∈ zorn-relation›
        unfolding zorn-relation-def order-double-subsets-def
        using Cmax-in-A by auto
      then have ‹Cmax = ((fst Cmax) ∪ {C0},snd Cmax)›
        using Cmax-is-max fst-extended-Cmax-in-A by fastforce
      then have ‹C0 ∈ (fst Cmax)›
        by (metis UnI2 fst-eqD singleton-iff)
      then show False using C0-def by auto
    qed
     moreover have snd-max-entailment-extended : ‹fst Cmax ⊨ snd Cmax ∪
{C0}›
    proof (rule ccontr)
      assume ‹¬fst Cmax ⊨ snd Cmax ∪ {C0}›
      then have snd-extended-Cmax-in-A : ‹(fst Cmax,snd Cmax ∪ {C0}) ∈ A›
        unfolding A-def
        using M-less-fst-Cmax N-less-snd-Cmax by blast
      then have ‹(Cmax,(fst Cmax,snd Cmax ∪ {C0})) ∈ zorn-relation›
        unfolding zorn-relation-def order-double-subsets-def
        using Cmax-in-A by auto
      then have ‹Cmax = (fst Cmax,snd Cmax ∪ {C0})›
        using Cmax-is-max snd-extended-Cmax-in-A by fastforce
      then have ‹C0 ∈ (snd Cmax)›
        by (metis UnI2 singleton-iff sndI)
      then show False using C0-def by auto
    qed
    ultimately have ‹fst Cmax ⊨ snd Cmax›
      using entails-cut by force
    then have ‹Cmax ∉ A›
      unfolding A-def
      by fastforce
```

```
    then show False
      using Cmax-in-A by simp
  qed
  then show ?thesis using M-less-fst-Cmax N-less-snd-Cmax Cmax-not-entails
by auto
 qed
 then show False using hyp-entails-sup by auto
qed


lemma entails-each: M ⊨ P ⟹ ∀ C∈M. N ⊨ Q ∪ {C} ⟹ ∀ D∈P. N ∪ {D}
⊨ Q ⟹ N ⊨ Q
proof −
 fix M P N Q
 assume m-entails-p: ‹M ⊨ P›
  and n-to-q-m: ‹∀ C∈M. N ⊨ Q ∪ {C}›
  and n-p-to-q: ‹∀ D∈P. N ∪ {D} ⊨ Q›
 have ‹N ⊆ M' ⟹ Q ⊆ N' ⟹ M' ∪ N' = UNIV ⟹ M' ⊨ N'› for M' N'
 proof −
  fix M' N'
  assume n-sub-mp: ‹M' ⊇ N› and
    q-sub-np: ‹N' ⊇ Q› and
    union-univ: ‹M' ∪ N' = UNIV›
  consider (a) ¬ (M' ∩ P = {}) | (b) ¬ (N' ∩ M = {}) | (c) P ⊆ N' ∧ M ⊆
M'
    using union-univ by auto
  then show ‹M' ⊨ N'›
  proof cases
    case a
    assume ‹M' ∩ P ≠ {}›
    then obtain D where d-in: ‹D ∈ M' ∩ P› by auto
    then have ‹N ∪ {D} ⊆ M'› using n-sub-mp by auto
    moreover have ‹N ∪ {D} ⊨ Q› using n-p-to-q d-in by blast
    ultimately show ?thesis
      using entails-subsets[OF - q-sub-np] by blast
  next
    case b
    assume ‹N' ∩ M ≠ {}›
    then obtain C where c-in: ‹C ∈ M ∩ N'› by auto
    then have ‹Q ∪ {C} ⊆ N'› using q-sub-np by auto
    moreover have ‹N ⊨ Q ∪ {C}› using n-to-q-m c-in by blast
    ultimately show ?thesis
      using entails-subsets[OF n-sub-mp] by blast
  next
    case c
    then show ?thesis
      using entails-subsets[OF - - m-entails-p] by simp
  qed
```

```
  qed
  then show ‹N ⊨ Q›
    using entails-supsets by simp
qed


lemma entails-bot-to-entails-empty: ‹{} ⊨ {bot} ⟹ {} ⊨ {}›
  using entails-reflexive[of bot] entails-each[of {} {bot} {} {}] bot-entails-empty
  by auto

abbreviation equi-entails :: 'f set ⇒ 'f set ⇒ bool where
  equi-entails M N ≡ (M ⊨ N ∧ N ⊨ M)

lemma entails-cond-reflexive: ‹N ≠ {} ⟹ N ⊨ N›
  using entails-reflexive entails-subsets by (meson bot.extremum from-nat-into in-
sert-subset)


lemma entails-empty-reflexive-dangerous: ‹{} ⊨ {} ⟹ M ⊨ N›
  using entails-subsets[of {} M {} N] by simp

definition entails-conjunctive :: 'f set ⇒ 'f set ⇒ bool (infix ⊨∩ 50) where
  M ⊨∩ N ≡ ∀ C∈N. M ⊨ {C}

sublocale Calculus.consequence-relation {bot} (⊨∩)
proof
  show {bot} ≠ {} by simp
next
  fix B N
  assume b-in: B ∈ {bot}
  then have b-is: B = bot by simp
  show {B} ⊨∩ N
    unfolding entails-conjunctive-def
    using entails-subsets[of {B} {B} {}] b-is bot-entails-empty by blast
next
  fix M N
  assume m-subs: (M :: 'f set) ⊆ N
  show ‹N ⊨∩ M› unfolding entails-conjunctive-def
  proof
    fix C
    assume C ∈ M
    then have c-subs: ‹{C} ⊆ N› using m-subs by fast
    show ‹N ⊨ {C}› using entails-subsets[OF c-subs - entails-reflexive[of C]] by
simp
  qed
next
  fix M N
  assume ‹∀ C∈M. N ⊨∩ {C}›
  then show ‹N ⊨∩ M›
```

```
    unfolding entails-conjunctive-def by blast
next
  fix M N P
  assume
    trans1: ‹M ⊨∩ N› and
    trans2: ‹N ⊨∩ P›
  show ‹M ⊨∩ P› unfolding entails-conjunctive-def
  proof
    fix C
    assume ‹C ∈ P›
   then have n-to-c: ‹N ⊨ {C}› using trans2 unfolding entails-conjunctive-def
by simp
    have M ∪ {C} ⊨ {C}
      using entails-subsets[OF - - entails-reflexive[of C], of M ∪ {C} {C}] by fast
    then have m-c-to-c: ‹∀ D∈{C}. M ∪ {D} ⊨ {C}› by blast
    have m-to-c-n: ∀ D∈N. M ⊨ {C} ∪ {D}
      using trans1 entails-subsets[of M M] unfolding entails-conjunctive-def by
blast
    show ‹M ⊨ {C}›
    using entails-each[OF n-to-c m-to-c-n m-c-to-c] unfolding entails-conjunctive-def
.
  qed
qed
end
```

3 Extension to Negated Formulas

```
datatype 'a sign = Pos 'a | Neg 'a

instance sign :: (countable) countable
  by (rule countable-classI [of (λx. case x of Pos x ⇒ to-nat (True, to-nat x)
                                             | Neg x ⇒ to-nat (False, to-nat x))])
   (smt (verit, best) Pair-inject from-nat-to-nat sign.exhaust sign.simps(5) sign.simps(6))

fun neg :: ‹'a sign ⇒ 'a sign› where
  ‹neg (Pos C) = Neg C› |
  ‹neg (Neg C) = Pos C›

fun to-V :: 'a sign ⇒ 'a where
  to-V (Pos C) = C |
  to-V (Neg C) = C

lemma neg-neg-A-is-A [simp]: ‹neg (neg A) = A›
  by (metis neg.simps(1) neg.simps(2) to-V.elims)

fun is-Pos :: 'a sign ⇒ bool where
  is-Pos (Pos C) = True |
  is-Pos (Neg C) = False
```

lemma *is-Pos-to-V*: ‹*is-Pos C* ⟹ *C* = *Pos* (*to-V C*)›
by (*metis is-Pos.simps*(*2*) *to-V.elims*)

lemma *is-Neg-to-V*: ‹¬ *is-Pos C* ⟹ *C* = *Neg* (*to-V C*)›
by (*metis is-Pos.simps*(*1*) *to-V.elims*)

lemma *pos-union-singleton*: ‹{*D*. *Pos D* ∈ *N* ∪ {*Pos X*}} = {*D*. *Pos D* ∈ *N*} ∪ {*X*}›
by *blast*

lemma *tov-set*[*simp*]: ‹{*to-V C* |*C*. *to-V C* ∈ *A*} = *A*›
by (*smt* (*verit*, *del-insts*) *mem-Collect-eq subsetI subset-antisym to-V.simps*(*1*))

lemma *pos-neg-union*: ‹{*P C* |*C*. *Q C* ∧ *is-Pos C*} ∪ {*P C* |*C*. *Q C* ∧ ¬ *is-Pos C*} = {*P C* |*C*. *Q C*}›
by *blast*

context *consequence-relation*
begin
definition *entails-neg* :: *'f sign set* ⇒ *'f sign set* ⇒ *bool* (**infix** ⊨∼ *50*) **where**
entails-neg M N ≡ {*C*. *Pos C* ∈ *M*} ∪ {*C*. *Neg C* ∈ *N*} ⊨ {*C*. *Pos C* ∈ *N*} ∪ {*C*. *Neg C* ∈ *M*}

lemma *swap-neg-in-entails-neg*: ‹{*neg A*} ⊨∼ {*neg B*} ⟷ {*B*} ⊨∼ {*A*}›
unfolding *entails-neg-def*
by (*smt* (*verit*, *ccfv-threshold*) *Collect-cong Un-commute mem-Collect-eq neg.simps*(*1*) *neg-neg-A-is-A singleton-conv2*)

lemma *ext-cons-rel*: ‹*consequence-relation* (*Pos bot*) *entails-neg*›
proof
show *entails-neg* {*Pos bot*} {}
unfolding *entails-neg-def* **using** *bot-entails-empty* **by** *simp*
next
fix *C*
show ‹*entails-neg* {*C*} {*C*}›
unfolding *entails-neg-def* **using** *entails-cond-reflexive*
by (*metis* (*mono-tags*, *lifting*) *Un-empty empty-Collect-eq insert-iff is-Pos.cases*)

next
fix *M N P Q*
assume
subs1: *M* ⊆ *N* **and**
subs2: *P* ⊆ *Q* **and**
entails1: *entails-neg M P*
have *union-subs1*: ‹{*C*. *Pos C* ∈ *M*} ∪ {*C*. *Neg C* ∈ *P*} ⊆ {*C*. *Pos C* ∈ *N*} ∪ {*C*. *Neg C* ∈ *Q*}›
using *subs1 subs2* **by** *auto*
have *union-subs2*: ‹{*C*. *Pos C* ∈ *P*} ∪ {*C*. *Neg C* ∈ *M*} ⊆ {*C*. *Pos C* ∈ *Q*} ∪ {*C*. *Neg C* ∈ *N*}›

using *subs1 subs2* **by** *auto*
have *union-entails1*: $\{C.\ Pos\ C \in M\} \cup \{C.\ Neg\ C \in P\} \models \{C.\ Pos\ C \in P\} \cup \{C.\ Neg\ C \in M\}$
using *entails1* **unfolding** *entails-neg-def* **.**
show ‹*entails-neg N Q*›
using *entails-subsets*[*OF union-subs1 union-subs2 union-entails1*] **unfolding** *entails-neg-def* **.**
next
fix *M N C M′ N′*
assume *cut-hypothesis-M-N*: ‹$M \models_\sim N \cup \{C\}$› **and**
cut-hypothesis-M′-N′: ‹$M' \cup \{C\} \models_\sim N'$›
consider (*a*) ‹*is-Pos C*› | (*b*) ‹¬*is-Pos C*›
by *auto*
then show ‹$M \cup M' \models_\sim N \cup N'$›
proof (*cases*)
case *a*
assume *Neg-C*: ‹*is-Pos C*›
have *M-entails-NC*:
‹$\{D.\ Pos\ D \in M\} \cup \{D.\ Neg\ D \in N \cup \{C\}\} \models \{D.\ Pos\ D \in N \cup \{C\}\} \cup \{D.\ Neg\ D \in M\}$›
using *cut-hypothesis-M-N entails-neg-def* **by** *force*
moreover have ‹$\{D.\ Pos\ D \in M\} \cup \{D.\ Neg\ D \in N \cup \{C\}\} = \{D.\ Pos\ D \in M\} \cup \{D.\ Neg\ D \in N\}$›
using *Neg-C* **by** *force*
ultimately have ‹$\{D.\ Pos\ D \in M\} \cup \{D.\ Neg\ D \in N\} \models \{D.\ Pos\ D \in N \cup \{C\}\} \cup \{D.\ Neg\ D \in M\}$›
by *simp*
moreover have ‹$\{D.\ Pos\ D \in N \cup \{C\}\} \cup \{D.\ Neg\ D \in M\} = \{D.\ Pos\ D \in N\} \cup \{to\text{-}V\ C\} \cup \{D.\ Neg\ D \in M\}$›
using *is-Pos-to-V*[*OF Neg-C*] **by** *force*
ultimately have *M-entails-NC-reformulated*:
‹$\{D.\ Pos\ D \in M\} \cup \{D.\ Neg\ D \in N\} \models \{D.\ Pos\ D \in N\} \cup \{to\text{-}V\ C\} \cup \{D.\ Neg\ D \in M\}$›
by *simp*
have *M′-entails-N′C*:
‹$\{D.\ Pos\ D \in M' \cup \{C\}\} \cup \{D.\ Neg\ D \in N'\} \models \{D.\ Pos\ D \in N'\} \cup \{D.\ Neg\ D \in M' \cup \{C\}\}$›
using *cut-hypothesis-M′-N′ entails-neg-def* **by** *force*
moreover have ‹$\{D.\ Pos\ D \in M' \cup \{C\}\} \cup \{D.\ Neg\ D \in N'\} = \{D.\ Pos\ D \in M'\} \cup \{to\text{-}V\ C\} \cup \{D.\ Neg\ D \in N'\}$›
using *is-Pos-to-V*[*OF Neg-C*] **by** *force*
ultimately have ‹$\{D.\ Pos\ D \in M'\} \cup \{to\text{-}V\ C\} \cup \{D.\ Neg\ D \in N'\} \models \{D.\ Pos\ D \in N'\} \cup \{D.\ Neg\ D \in M' \cup \{C\}\}$›
by *simp*
moreover have ‹$\{D.\ Pos\ D \in N'\} \cup \{D.\ Neg\ D \in M' \cup \{C\}\} = \{D.\ Pos\ D \in N'\} \cup \{D.\ Neg\ D \in M'\}$›
using *Neg-C* **by** *force*
ultimately have *M′-entails-N′C-reformulated*:
‹$\{D.\ Pos\ D \in M'\} \cup \{to\text{-}V\ C\} \cup \{D.\ Neg\ D \in N'\} \models \{D.\ Pos\ D \in N'\}$

$\cup \{D.\ Neg\ D \in M'\}\rangle$
by *simp*
have $\langle\{D.\ Pos\ D \in M\} \cup \{D.\ Neg\ D \in N\} \cup \{D.\ Pos\ D \in M'\} \cup \{D.\ Neg\ D \in N'\} \models$
$\{D.\ Pos\ D \in N\} \cup \{D.\ Neg\ D \in M\} \cup \{D.\ Pos\ D \in N'\} \cup \{D.\ Neg\ D \in M'\}\rangle$
using *M-entails-NC-reformulated M′-entails-N′C-reformulated entails-cut*
by (*smt* (*verit*, *ccfv-threshold*) *M′-entails-N′C-reformulated M-entails-NC-reformulated Un-assoc Un-commute*)
moreover have $\langle\{D.\ Pos\ D \in M\} \cup \{D.\ Neg\ D \in N\} \cup \{D.\ Pos\ D \in M'\} \cup \{D.\ Neg\ D \in N'\} =$
$\{D.\ Pos\ D \in M \cup M'\} \cup \{D.\ Neg\ D \in N \cup N'\}\rangle$
by *auto*
ultimately have $\langle\{D.\ Pos\ D \in M \cup M'\} \cup \{D.\ Neg\ D \in N \cup N'\} \models$
$\{D.\ Pos\ D \in N\} \cup \{D.\ Neg\ D \in M\} \cup \{D.\ Pos\ D \in N'\} \cup \{D.\ Neg\ D \in M'\}\rangle$
by *simp*
moreover have $\langle\{D.\ Pos\ D \in N\} \cup \{D.\ Neg\ D \in M\} \cup \{D.\ Pos\ D \in N'\} \cup \{D.\ Neg\ D \in M'\} =$
$\{D.\ Pos\ D \in N \cup N'\} \cup \{D.\ Neg\ D \in M \cup M'\}\rangle$
by *auto*
ultimately have $\langle\{D.\ Pos\ D \in M \cup M'\} \cup \{D.\ Neg\ D \in N \cup N'\} \models$
$\{D.\ Pos\ D \in N \cup N'\} \cup \{D.\ Neg\ D \in M \cup M'\}\rangle$
by *simp*
then show *?thesis* **unfolding** *entails-neg-def* **by** *auto*
next
case *b*
assume *Neg-C*: $\langle\neg$*is-Pos C*$\rangle$
have *M-entails-NC*:
$\langle\{D.\ Pos\ D \in M\} \cup \{D.\ Neg\ D \in N \cup \{C\}\} \models \{D.\ Pos\ D \in N \cup \{C\}\} \cup \{D.\ Neg\ D \in M\}\rangle$
using *cut-hypothesis-M-N entails-neg-def* **by** *force*
moreover have $\langle\{D.\ Pos\ D \in M\} \cup \{D.\ Neg\ D \in N \cup \{C\}\} =$
$\{D.\ Pos\ D \in M\} \cup \{to\text{-}V\ C\} \cup \{D.\ Neg\ D \in N\}\rangle$
using *is-Neg-to-V*[*OF Neg-C*] **by** *force*
ultimately have $\langle\{D.\ Pos\ D \in M\} \cup \{to\text{-}V\ C\} \cup \{D.\ Neg\ D \in N\} \models$
$\{D.\ Pos\ D \in N \cup \{C\}\} \cup \{D.\ Neg\ D \in M\}\rangle$
by *simp*
moreover have $\langle\{D.\ Pos\ D \in N \cup \{C\}\} \cup \{D.\ Neg\ D \in M\} = \{D.\ Pos\ D \in N\} \cup \{D.\ Neg\ D \in M\}\rangle$
using *Neg-C* **by** *force*
ultimately have *M-entails-NC-reformulated*:
$\langle\{D.\ Pos\ D \in M\} \cup \{to\text{-}V\ C\} \cup \{D.\ Neg\ D \in N\} \models \{D.\ Pos\ D \in N\} \cup \{D.\ Neg\ D \in M\}\rangle$
by *simp*
have *M′-entails-N′C*:
$\langle\{D.\ Pos\ D \in M' \cup \{C\}\} \cup \{D.\ Neg\ D \in N'\} \models \{D.\ Pos\ D \in N'\} \cup \{D.\ Neg\ D \in M' \cup \{C\}\}\rangle$
using *cut-hypothesis-M′-N′ entails-neg-def* **by** *force*

moreover have ‹{*D*. *Pos D* ∈ *M′* ∪ {*C*}} ∪ {*D*. *Neg D* ∈ *N′*} = {*D*. *Pos D* ∈ *M′*} ∪ {*D*. *Neg D* ∈ *N′*}›
using *Neg-C* **by** *force*
ultimately have ‹{*D*. *Pos D* ∈ *M′*} ∪ {*D*. *Neg D* ∈ *N′*} ⊨ {*D*. *Pos D* ∈ *N′*} ∪ {*D*. *Neg D* ∈ *M′* ∪ {*C*}}›
by *simp*
moreover have ‹{*D*. *Pos D* ∈ *N′*} ∪ {*D*. *Neg D* ∈ *M′* ∪ {*C*}} =
{*D*. *Pos D* ∈ *N′*} ∪ {*to-V C*} ∪ {*D*. *Neg D* ∈ *M′*}›
using *is-Neg-to-V*[*OF Neg-C*] **by** *force*
ultimately have *M′-entails-N′C-reformulated*:
‹{*D*. *Pos D* ∈ *M′*} ∪ {*D*. *Neg D* ∈ *N′*} ⊨ {*D*. *Pos D* ∈ *N′*} ∪ {*to-V C*} ∪ {*D*. *Neg D* ∈ *M′*}›
by *simp*
have ‹{*D*. *Pos D* ∈ *M*} ∪ {*D*. *Neg D* ∈ *N*} ∪ {*D*. *Pos D* ∈ *M′*} ∪ {*D*. *Neg D* ∈ *N′*}⊨
{*D*. *Pos D* ∈ *N*} ∪ {*D*. *Neg D* ∈ *M*} ∪ {*D*. *Pos D* ∈ *N′*} ∪ {*D*. *Neg D* ∈ *M′*}›
using *M-entails-NC-reformulated M′-entails-N′C-reformulated entails-cut*
by (*smt* (*verit*, *ccfv-threshold*) *M′-entails-N′C-reformulated*
M-entails-NC-reformulated Un-assoc Un-commute)
moreover have ‹{*D*. *Pos D* ∈ *M*} ∪ {*D*. *Neg D* ∈ *N*} ∪ {*D*. *Pos D* ∈ *M′*} ∪ {*D*. *Neg D* ∈ *N′*} =
{*D*. *Pos D* ∈ *M* ∪ *M′*} ∪ {*D*. *Neg D* ∈ *N* ∪ *N′*}›
by *auto*
ultimately have ‹{*D*. *Pos D* ∈ *M* ∪ *M′*} ∪ {*D*. *Neg D* ∈ *N* ∪ *N′*} ⊨
{*D*. *Pos D* ∈ *N*} ∪ {*D*. *Neg D* ∈ *M*} ∪ {*D*. *Pos D* ∈ *N′*} ∪ {*D*. *Neg D* ∈ *M′*}›
by *simp*
moreover have ‹{*D*. *Pos D* ∈ *N*} ∪ {*D*. *Neg D* ∈ *M*} ∪ {*D*. *Pos D* ∈ *N′*} ∪ {*D*. *Neg D* ∈ *M′*} =
{*D*. *Pos D* ∈ *N* ∪ *N′*} ∪ {*D*. *Neg D* ∈ *M* ∪ *M′*}›
by *auto*
ultimately have ‹{*D*. *Pos D* ∈ *M* ∪ *M′*} ∪ {*D*. *Neg D* ∈ *N* ∪ *N′*} ⊨
{*D*. *Pos D* ∈ *N* ∪ *N′*} ∪ {*D*. *Neg D* ∈ *M* ∪ *M′*}›
by *simp*
then show *?thesis* **unfolding** *entails-neg-def* **by** *auto*
qed
next
fix *M N*
assume *M-entails-N*: ‹*M* ⊨∼ *N*›
then have ‹{*C*. *Pos C* ∈ *M*} ∪ {*C*. *Neg C* ∈ *N*} ⊨ {*C*. *Pos C* ∈ *N*} ∪ {*C*. *Neg C* ∈ *M*}›
unfolding *entails-neg-def* **.**
then have ‹∃ *M′ N′*. (*M′* ⊆ {*C*. *Pos C* ∈ *M*} ∪ {*C*. *Neg C* ∈ *N*} ∧
N′ ⊆ {*C*. *Pos C* ∈ *N*} ∪ {*C*. *Neg C* ∈ *M*} ∧
finite M′ ∧ *finite N′* ∧ *M′* ⊨ *N′*)›
using *entails-compactness* **by** *auto*
then obtain *M′ N′* **where**
M′-def: ‹*M′* ⊆ {*C*. *Pos C* ∈ *M*} ∪ {*C*. *Neg C* ∈ *N*}› **and**

M'-*finite*: ‹*finite* M'› **and**
N'-*def*: ‹$N' \subseteq \{C.\ Pos\ C \in N\} \cup \{C.\ Neg\ C \in M\}$› **and**
N'-*finite*: ‹*finite* N'› **and**
M'-*entails-N'*: ‹$M' \models N'$›
by *auto*
define M'-*pos* **where** M'-*pos* $= M' \cap \{C.\ Pos\ C \in M\}$
define M'-*neg* **where** M'-*neg* $= M' \cap \{C.\ Neg\ C \in N\}$
define N'-*pos* **where** N'-*pos* $= N' \cap \{C.\ Pos\ C \in N\}$
define N'-*neg* **where** N'-*neg* $= N' \cap \{C.\ Neg\ C \in M\}$
have *compactness-hypothesis*:
‹M'-*pos* $\cup$ M'-*neg* $\models$ N'-*pos* $\cup$ N'-*neg*›
using *inf.absorb-iff1 inf-sup-distrib1 M'-def N'-def M'-entails-N'*
unfolding M'-*pos-def* M'-*neg-def* N'-*pos-def* N'-*neg-def*
by (*smt* (*verit*))
have M'-*pos-finite*: ‹*finite* M'-*pos*› **using** M'-*finite* **unfolding** M'-*pos-def* **by** *blast*
have ‹*finite* M'-*neg*› **using** M'-*finite* **unfolding** M'-*neg-def* **by** *blast*
have ‹*finite* N'-*pos*› **using** N'-*finite* **unfolding** N'-*pos-def* **by** *blast*
have ‹*finite* N'-*neg*› **using** N'-*finite* **unfolding** N'-*neg-def* **by** *blast*
define *M-fin* **where** *M-fin* $= \{Pos\ C\ |C.\ Pos\ C \in M \wedge C \in M'\} \cup \{Neg\ C\ |C.\ Neg\ C \in M \wedge C \in N'\}$
then have *fin-M-fin*: ‹*finite M-fin*› **using** M'-*finite* N'-*finite* **by** *auto*
have *sub-M-fin*: ‹*M-fin* $\subseteq$ M›
unfolding *M-fin-def* **by** *blast*
define *N-fin* **where** *N-fin* $= \{Pos\ C\ |C.\ Pos\ C \in N \wedge C \in N'\} \cup \{Neg\ C\ |C.\ Neg\ C \in N \wedge C \in M'\}$
then have *fin-N-fin*: ‹*finite N-fin*› **using** N'-*finite* M'-*finite* **by** *auto*
have *sub-N-fin*: ‹*N-fin* $\subseteq$ N›
unfolding *N-fin-def* **by** *blast*
have ‹$\{C.\ Pos\ C \in$ *M-fin*$\}$ = M'-*pos*›
unfolding *M-fin-def* M'-*pos-def* **by** *blast*
moreover have ‹$\{C.\ Neg\ C \in$ *M-fin*$\}$ = N'-*neg*›
unfolding *M-fin-def* N'-*neg-def* **by** *blast*
moreover have ‹$\{C.\ Pos\ C \in$ *N-fin*$\}$ = N'-*pos*›
unfolding *N-fin-def* N'-*pos-def* **by** *blast*
moreover have ‹$\{C.\ Neg\ C \in$ *N-fin*$\}$ = M'-*neg*›
unfolding *N-fin-def* M'-*neg-def* **by** *blast*
ultimately have ‹*M-fin* $\models_\sim$ *N-fin*›
unfolding *entails-neg-def* **using** *compactness-hypothesis* **by** *force*
then show ‹$\exists\, M'\ N'.\ M' \subseteq M \wedge N' \subseteq N \wedge$ *finite* $M' \wedge$ *finite* $N' \wedge M' \models_\sim N'$›
using *fin-M-fin fin-N-fin sub-M-fin sub-N-fin* **by** *auto*
qed

interpretation *neg-ext-cons-rel*: *consequence-relation Pos bot entails-neg*
using *ext-cons-rel* **by** *simp*

lemma *pos-neg-entails-bot*: ‹$\{C\} \cup \{neg\ C\} \models_\sim \{Pos\ bot\}$›
proof −

have ‹$\{C\} \cup \{neg\ C\} \models_\sim \{\}$› **unfolding** *entails-neg-def*
by (*smt* (*verit*, *del-insts*) *Un-iff empty-subsetI entails-reflexive entails-subsets insertI1*
insert-is-Un insert-subset is-Pos.cases mem-Collect-eq neg.simps(*1*) *neg.simps*(*2*))
then show *?thesis* **using** *neg-ext-cons-rel.entails-subsets* **by** *blast*
qed

lemma *entails-of-entails-iff*:
‹$\{C\} \models_\sim Cs \Longrightarrow$ *finite* $Cs \Longrightarrow$ *card* $Cs \geq 1 \Longrightarrow$
$(\forall\ C_i \in Cs.\ \mathcal{M} \cup \{C_i\} \models_\sim \{Pos\ bot\}) \Longrightarrow \mathcal{M} \cup \{C\} \models_\sim \{Pos\ bot\}$›
proof −
assume ‹$\{C\} \models_\sim Cs$› **and**
finite-Cs: ‹*finite* Cs› **and**
Cs-not-empty: ‹*card* $Cs \geq 1$› **and**
all-C_i*-entail-bot*: ‹$\forall\ C_i \in Cs.\ \mathcal{M} \cup \{C_i\} \models_\sim \{Pos\ bot\}$›
then have ‹$\mathcal{M} \cup \{C\} \models_\sim Cs$›
using *Un-upper2 consequence-relation.entails-subsets subsetI*
by (*metis neg-ext-cons-rel.entails-subsets*)
then show ‹$\mathcal{M} \cup \{C\} \models_\sim \{Pos\ bot\}$›
using *Cs-not-empty all-*C_i*-entail-bot*
proof (*induct rule*: *finite-ne-induct*[*OF finite-Cs*])
case *1*
then show *?case*
using *Cs-not-empty*
by *force*
next
case (*2 x*)
then show *?case*
using *consequence-relation.entails-cut ext-cons-rel*
by *fastforce*
next
case *insert*: (*3 x F*)

have *card-F-ge-1*: ‹*card* $F \geq 1$›
by (*meson card-0-eq insert.hyps*(*1*) *insert.hyps*(*2*) *less-one linorder-not-less*)

have ‹$\mathcal{M} \cup \{C\} \models_\sim \{Pos\ bot, x\} \cup F$›
by (*smt* (*verit*, *ccfv-threshold*) *Un-insert-left Un-insert-right Un-upper2*
consequence-relation.entails-subsets insert.prems(*1*) *insert-is-Un ext-cons-rel*)
then have ‹$\mathcal{M} \cup \{C\} \models_\sim \{Pos\ bot\} \cup \{x\} \cup F$›
by (*metis insert-is-Un*)
moreover have ‹$\mathcal{M} \cup \{C\} \cup \{x\} \models_\sim \{Pos\ bot\} \cup F$›
by (*smt* (*verit*, *ccfv-SIG*) *Un-upper2 consequence-relation.entails-subsets insert.prems*(*3*)
insertCI ext-cons-rel sup-assoc sup-ge1 sup-left-commute)
ultimately have ‹$\mathcal{M} \cup \{C\} \models_\sim \{Pos\ bot\} \cup F$›
using *consequence-relation.entails-cut ext-cons-rel*
by *fastforce*
then have ‹$\mathcal{M} \cup \{C\} \models_\sim F$›

using *consequence-relation.bot-entails-empty consequence-relation.entails-cut ext-cons-rel*
by *fastforce*
then show *?case*
using *insert card-F-ge-1*
by *blast*
qed
qed

end

end

theory *Calculi-And-Annotations*
imports *Disjunctive-Consequence-Relations*
begin

4 Calculi and Derivations

context
begin

no-notation *Extended.extended.Pinf* (‹∞›)

typedef *'a infinite-llist* = ‹{ *l* :: *'a llist*. *llength l* = ∞ }›
morphisms *to-llist Abs-infinite-llist*
using *llength-inf-llist*
by *blast*

setup-lifting *type-definition-infinite-llist*

lift-definition *llmap* :: ‹(*'a* ⇒ *'b*) ⇒ *'a infinite-llist* ⇒ *'b infinite-llist*› **is** *lmap*
by *auto*

lift-definition *llnth* :: ‹*'a infinite-llist* ⇒ *nat* ⇒ *'a*› **is** *lnth* **.**

lift-definition *Liminf-infinite-llist* :: ‹*'a set infinite-llist* ⇒ *'a set*› **is** *Liminf-llist*
.

lift-definition *Limsup-infinite-llist* :: ‹*'a set infinite-llist* ⇒ *'a set*› **is** *Limsup-llist*
.

lift-definition *Sup-infinite-llist* :: ‹*'a set infinite-llist* ⇒ *'a set*› **is** *Sup-llist* **.**

lift-definition *llhd* :: ‹*'a infinite-llist* ⇒ *'a*› **is** *lhd* **.**

lemma *llength-of-to-llist-is-infinite*: ‹*llength* (*to-llist L*) = ∞›

using *to-llist*
by *auto*

end

locale *sound-inference-system* =
inference-system Inf + *sound-cons*: *consequence-relation bot entails-sound*
for
Inf :: *'f inference set* **and**
bot :: *'f* **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨*s* *50*)
+ **assumes**
sound: *ι* ∈ *Inf* ⟹ *set* (*prems-of ι*) ⊨*s* {*concl-of ι*}

no-notation *IArray.sub* (**infixl** !! *100*)

definition *is-derivation* :: (*'f set* ⇒ *'f set* ⇒ *bool*) ⇒ (*'f set infinite-llist*) ⇒ *bool*
where
is-derivation R Ns ≡ ∀ *i*. *R* (*llnth Ns i*) (*llnth Ns* (*Suc i*))

definition *terminates* :: *'f set infinite-llist* ⇒ *bool* **where**
terminates Ns ≡ ∃ *i*. ∀ *j*>*i*. *llnth Ns j* = *llnth Ns i*

abbreviation ‹*lim-inf* ≡ *Liminf-infinite-llist*›

abbreviation *limit* :: *'f set infinite-llist* ⇒ *'f set* **where** *limit Ns* ≡ *lim-inf Ns*

abbreviation *lim-sup* :: ‹*'f set infinite-llist* ⇒ *'f set*› **where**
‹*lim-sup Ns* ≡ *Limsup-infinite-llist Ns*›

locale *calculus* = *inference-system Inf* + *consequence-relation bot entails*
for
bot :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨ *50*)
+ **fixes**
Red_I :: *'f set* ⇒ *'f inference set* **and**
Red_F :: *'f set* ⇒ *'f set*
assumes
Red-I-to-Inf: $Red_I\ N \subseteq Inf$ **and**
Red-F-Bot: $N \models \{bot\} \Longrightarrow N - Red_F\ N \models \{bot\}$ **and**
Red-F-of-subset: $N \subseteq N' \Longrightarrow Red_F\ N \subseteq Red_F\ N'$ **and**
Red-I-of-subset: $N \subseteq N' \Longrightarrow Red_I\ N \subseteq Red_I\ N'$ **and**
Red-F-of-Red-F-subset: $N' \subseteq Red_F\ N \Longrightarrow Red_F\ N \subseteq Red_F\ (N - N')$ **and**
Red-I-of-Red-F-subset: $N' \subseteq Red_F\ N \Longrightarrow Red_I\ N \subseteq Red_I\ (N - N')$ **and**
Red-I-of-Inf-to-N: *ι* ∈ *Inf* ⟹ *concl-of ι* ∈ *N* ⟹ *ι* ∈ $Red_I\ N$
begin

definition *saturated* :: *'f set* ⇒ *bool* **where**

saturated $N \longleftrightarrow$ *Inf-from* $N \subseteq Red_I\ N$

definition Red_I*-strict* :: *'f set* $\Rightarrow$ *'f inference set* **where**
Red_I*-strict* $N = \{\iota.\ \iota \in Red_I\ N \vee (\iota \in Inf \wedge bot \in N)\}$

definition Red_F*-strict* :: *'f set* $\Rightarrow$ *'f set* **where**
Red_F*-strict* $N = \{C.\ C \in Red_F\ N \vee (bot \in N \wedge C \neq bot)\}$

lemma *strict-calc-if-nobot*:
$\forall N.\ bot \notin Red_F\ N \Longrightarrow$ *calculus bot Inf entails* Red_I*-strict* Red_F*-strict*
proof
fix N
show ‹Red_I*-strict* $N \subseteq$ *Inf*› **unfolding** Red_I*-strict-def* **using** *Red-I-to-Inf* **by** *blast*
next
fix N
assume
bot-notin: $\forall N.\ bot \notin Red_F\ N$ **and**
entails-bot: ‹$N \models \{bot\}$›
show ‹$N - Red_F$*-strict* $N \models \{bot\}$›
proof (*cases* $bot \in N$)
assume *bot-in*: $bot \in N$
have ‹$bot \notin Red_F\ N$› **using** *bot-notin* **by** *blast*
then have ‹$bot \notin Red_F$*-strict* N› **unfolding** Red_F*-strict-def* **by** *blast*
then have ‹Red_F*-strict* $N = UNIV - \{bot\}$›
unfolding Red_F*-strict-def* **using** *bot-in* **by** *blast*
then have ‹$N - Red_F$*-strict* $N = \{bot\}$› **using** *bot-in* **by** *blast*
then show ‹$N - Red_F$*-strict* $N \models \{bot\}$› **using** *entails-reflexive*[*of bot*] **by** *simp*
next
assume ‹$bot \notin N$›
then have ‹Red_F*-strict* $N = Red_F\ N$› **unfolding** Red_F*-strict-def* **by** *blast*
then show ‹$N - Red_F$*-strict* $N \models \{bot\}$› **using** *Red-F-Bot*[*OF entails-bot*] **by** *simp*
qed
next
fix $N\ N'$ :: *'f set*
assume ‹$N \subseteq N'$›
then show ‹Red_F*-strict* $N \subseteq Red_F$*-strict* N'›
unfolding Red_F*-strict-def* **using** *Red-F-of-subset* **by** *blast*
next
fix $N\ N'$ :: *'f set*
assume ‹$N \subseteq N'$›
then show ‹Red_I*-strict* $N \subseteq Red_I$*-strict* N'›
unfolding Red_I*-strict-def* **using** *Red-I-of-subset* **by** *blast*
next
fix $N'\ N$
assume

```
    bot-notin: ∀ N. bot ∉ Red_F N and
    subs-red: N' ⊆ Red_F-strict N
  have ‹bot ∉ Red_F-strict N›
    using bot-notin unfolding Red_F-strict-def by blast
  then have nbot-in: ‹bot ∉ N'› using subs-red by blast
  show ‹Red_F-strict N ⊆ Red_F-strict (N − N')›
  proof (cases bot ∈ N)
    case True
    then have bot-in: bot ∈ N − N' using nbot-in by blast
    then show ?thesis unfolding Red_F-strict-def using bot-notin by force
  next
    case False
     then have eq-red: Red_F-strict N = Red_F N unfolding Red_F-strict-def by
simp
    then have N' ⊆ Red_F N using subs-red by simp
    then have Red_F N ⊆ Red_F (N − N') using Red-F-of-Red-F-subset by simp
    then show ?thesis using eq-red Red_F-strict-def by blast
  qed
next
  fix N' N
  assume
    ∀ N. bot ∉ Red_F N and
    subs-red: N' ⊆ Red_F-strict N
  then have bot-notin: bot ∉ N' unfolding Red_F-strict-def by blast
  then show Red_I-strict N ⊆ Red_I-strict (N − N')
  proof (cases bot ∈ N)
    case True
    then show ?thesis
      unfolding Red_I-strict-def using bot-notin Red-I-to-Inf by fastforce
  next
    case False
    then show ?thesis
      using bot-notin Red-I-to-Inf subs-red Red-I-of-Red-F-subset
      unfolding Red_I-strict-def Red_F-strict-def by simp
  qed
next
  fix ι N
  assume ι ∈ Inf
  then show concl-of ι ∈ N ⟹ ι ∈ Red_I-strict N
    unfolding Red_I-strict-def using Red-I-of-Inf-to-N Red-I-to-Inf by simp
qed

definition weakly-fair :: 'f set infinite-llist ⇒ bool where
  ‹weakly-fair Ns ≡ Inf-from (Liminf-infinite-llist Ns) ⊆ Sup-infinite-llist (llmap
Red_I Ns)›

abbreviation fair :: 'f set infinite-llist ⇒ bool where fair N ≡ weakly-fair N

definition derive :: 'f set ⇒ 'f set ⇒ bool (infix ▷ 50) where
```

$M \rhd N \equiv (M - N \subseteq Red_F\ N)$

lemma *derive-refl*: $M \rhd M$ **unfolding** *derive-def* **by** *simp*

lemma *deriv-red-in*: ‹$M \rhd N \Longrightarrow Red_F\ M \subseteq N \cup Red_F\ N$›
proof −
fix *M N*
assume *deriv*: ‹$M \rhd N$›
then have ‹$M \subseteq N \cup Red_F\ N$›
unfolding *derive-def* **by** *blast*
then have *red-m-in*: ‹$Red_F\ M \subseteq Red_F\ (N \cup Red_F\ N)$›
using *Red-F-of-subset* **by** *blast*
have ‹$Red_F\ (N \cup Red_F\ N) \subseteq Red_F\ (N \cup Red_F\ N - (Red_F\ N - N))$›
using *Red-F-of-Red-F-subset*[*of* $Red_F\ N - N\ N \cup Red_F\ N$]
Red-F-of-subset[*of* $N\ N \cup Red_F\ N$] **by** *fast*
then have ‹$Red_F\ (N \cup Red_F\ N) \subseteq Red_F\ N$›
by (*metis Diff-subset-conv Red-F-of-subset Un-Diff-cancel lfp.leq-trans subset-refl sup.commute*)
then show ‹$Red_F\ M \subseteq N \cup Red_F\ N$› **using** *red-m-in* **by** *blast*
qed

lemma *derive-trans*: $M \rhd N \Longrightarrow N \rhd N' \Longrightarrow M \rhd N'$
using *deriv-red-in* **by** (*smt* (*z3*) *Diff-subset-conv derive-def subset-trans sup.absorb-iff2*)

end

locale *sound-calculus* = *sound-inference-system Inf bot entails-sound* +
consequence-relation bot entails
for
bot :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** $\models$ *50*) **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** $\models s$ *50*)
+ **fixes**
Red_I :: *'f set* ⇒ *'f inference set* **and**
Red_F :: *'f set* ⇒ *'f set*
assumes
Red-I-to-Inf: $Red_I\ N \subseteq Inf$ **and**
Red-F-Bot: $N \models \{bot\} \Longrightarrow N - Red_F\ N \models \{bot\}$ **and**
Red-F-of-subset: $N \subseteq N' \Longrightarrow Red_F\ N \subseteq Red_F\ N'$ **and**
Red-I-of-subset: $N \subseteq N' \Longrightarrow Red_I\ N \subseteq Red_I\ N'$ **and**
Red-F-of-Red-F-subset: $N' \subseteq Red_F\ N \Longrightarrow Red_F\ N \subseteq Red_F\ (N - N')$ **and**
Red-I-of-Red-F-subset: $N' \subseteq Red_F\ N \Longrightarrow Red_I\ N \subseteq Red_I\ (N - N')$ **and**
Red-I-of-Inf-to-N: $\iota \in Inf \Longrightarrow concl\text{-}of\ \iota \in N \Longrightarrow \iota \in Red_I\ N$
begin

sublocale *calculus bot Inf entails*
  by (*simp add*: *calculus.intro calculus-axioms.intro Red-F-Bot*
    Red-F-of-Red-F-subset Red-F-of-subset Red-I-of-Inf-to-N Red-I-of-Red-F-subset
Red-I-of-subset
    Red-I-to-Inf consequence-relation-axioms)
end

locale *statically-complete-calculus* = *calculus* +
  assumes *statically-complete*: *saturated N* $\Longrightarrow$ *N* $\models$ $\{bot\}$ $\Longrightarrow$ *bot* $\in$ *N*
begin

lemma *inf-from-subs*: *M* $\subseteq$ *N* $\Longrightarrow$ *Inf-from M* $\subseteq$ *Inf-from N*
  unfolding *Inf-from-def* **by** *blast*

lemma *nobot-in-Red*: ‹*bot* $\notin$ Red_F *N*›
proof −
  have ‹*UNIV* $\models$ $\{bot\}$›
    using *entails-reflexive*[*of bot*] *entails-subsets*[*of* $\{bot\}$ *UNIV* $\{bot\}$ $\{bot\}$] **by**
fast
  then have *non-red-entails-bot*: ‹*UNIV* − (Red_F *UNIV*) $\models$ $\{bot\}$› **using** *Red-F-Bot*[*of*
UNIV] **by** *simp*
  have ‹*Inf-from UNIV* $\subseteq$ Red_I *UNIV*›
    unfolding *Inf-from-def* **using** *Red-I-of-Inf-to-N*[*of - UNIV*] **by** *blast*
  then have *sat-non-red*: ‹*saturated* (*UNIV* − Red_F *UNIV*)›
    unfolding *saturated-def Inf-from-def* **using** *Red-I-of-Red-F-subset*[*of* Red_F
UNIV UNIV] **by** *blast*
  have ‹*bot* $\notin$ Red_F *UNIV*›
    using *statically-complete*[*OF sat-non-red non-red-entails-bot*] **by** *fast*
  then show *?thesis* **using** *Red-F-of-subset*[*of - UNIV*] **by** *auto*
qed

interpretation *strict-calculus*:
  statically-complete-calculus bot Inf entails Red_I*-strict* Red_F*-strict*
proof −
  interpret *strict-calc*: *calculus bot Inf entails* Red_I*-strict* Red_F*-strict*
  using *strict-calc-if-nobot nobot-in-Red* **by** *blast*

  have ‹*saturated N* $\Longrightarrow$ *strict-calc.saturated N*›
    unfolding *saturated-def strict-calc.saturated-def* Red_I*-strict-def* **by** *blast*
  have ‹*strict-calc.saturated N* $\Longrightarrow$ *N* $\models$ $\{bot\}$ $\Longrightarrow$ *bot* $\in$ *N*› **for** *N*
  proof −
    assume
      strict-sat: *strict-calc.saturated N* **and**
      entails-bot: *N* $\models$ $\{bot\}$
    have ‹*bot* $\notin$ *N* $\Longrightarrow$ Red_I*-strict N* = Red_I *N*› **unfolding** Red_I*-strict-def* **by**
simp
    then have ‹*bot* $\notin$ *N* $\Longrightarrow$ *saturated N*›

unfolding *saturated-def* **using** *strict-sat* **by** (*simp add*: *strict-calc.saturated-def*)

then have ‹*bot* ∉ *N* ⟹ *bot* ∈ *N*›
using *statically-complete*[*OF - entails-bot*] **by** *simp*
then show ‹*bot* ∈ *N*› **by** *auto*
qed
then show ‹*statically-complete-calculus bot Inf entails* Red_I*-strict* Red_F*-strict*›
unfolding *statically-complete-calculus-def statically-complete-calculus-axioms-def*
using *strict-calc.calculus-axioms* **by** *blast*
qed

end

locale *dynamically-complete-calculus* = *calculus* +
assumes *dynamically-complete*:
‹*is-derivation* (▷) *Ns* ⟹ *fair Ns* ⟹ *llhd Ns* ⊨ {*bot*} ⟹ ∃ *i*. *bot* ∈ *llnth Ns* *i*›

5 Annotated Formulas and Consequence Relations

datatype ('*f*, '*v*::*countable*) *AF* = *Pair* (*F-of*: '*f*) (*A-of*: '*v sign fset*)

definition *is-interpretation* :: '*v sign set* ⇒ *bool* **where**
‹*is-interpretation J* = (∀ *v1*∈*J*. (∀ *v2*∈*J*. (*to-V v1* = *to-V v2* ⟶ *v1* = *v2*)))›

typedef '*v propositional-interpretation* = {*J* :: '*v sign set*. *is-interpretation J*}
proof
show ‹{} ∈ {*J*. *is-interpretation J*}› **unfolding** *is-interpretation-def* **by** *blast*
qed

abbreviation *interp-of* ≡ *Abs-propositional-interpretation*
abbreviation *strip* ≡ *Rep-propositional-interpretation*

setup-lifting *type-definition-propositional-interpretation*

lift-definition *belong-to* :: '*v sign* ⇒ '*v propositional-interpretation* ⇒ *bool* (**infix** ∈$_J$ *90*)
is (∈)::('*v sign* ⇒ '*v sign set* ⇒ *bool*) **.**

definition *total* :: '*v propositional-interpretation* ⇒ *bool* **where**
‹*total J* ≡ (∀ *v*. (∃ v_J. v_J ∈$_J$ *J* ∧ *to-V* v_J = *v*))›

typedef '*v total-interpretation* = {*J* :: '*v propositional-interpretation*. *total J*}
proof
show ‹*interp-of* (*Pos* ‘ (*UNIV* :: '*v set*)) ∈ {*J*. *total J*}›
unfolding *total-def*
proof
show ∀ *v*. ∃ v_J. v_J ∈$_J$ *interp-of* (*range Pos*) ∧ *to-V* v_J = *v*
proof

fix v
have $Pos\ v \in_J interp\text{-}of\ (range\ Pos) \wedge to\text{-}V\ (Pos\ v) = v$
by (*simp add*: *Abs-propositional-interpretation-inverse belong-to.rep-eq is-interpretation-def*)
then show $\exists\, v_J.\ v_J \in_J interp\text{-}of\ (range\ Pos) \wedge to\text{-}V\ v_J = v$ **by** *blast*
qed
qed
qed

abbreviation *total-interp-of* $\equiv (\lambda x.\ Abs\text{-}total\text{-}interpretation\ (interp\text{-}of\ x))$
abbreviation *total-strip* $\equiv (\lambda x.\ strip\ (Rep\text{-}total\text{-}interpretation\ x))$

lemma *neg-notin-total-strip* [*simp*]: $\langle(neg\ a \notin total\text{-}strip\ J) = (a \in total\text{-}strip\ J)\rangle$
proof
assume *neg-a-notin*: $\langle neg\ a \notin total\text{-}strip\ J\rangle$
have $\langle\exists\, b.\ to\text{-}V\ a = to\text{-}V\ b \wedge b \in total\text{-}strip\ J\rangle$
by (*metis Rep-total-interpretation belong-to.rep-eq mem-Collect-eq total-def*)
then show $\langle a \in total\text{-}strip\ J\rangle$
using *neg-a-notin* **by** (*metis neg.simps to-V.elims*)
next
assume *a-in*: $\langle a \in total\text{-}strip\ J\rangle$
then have $\langle\exists\, b.\ to\text{-}V\ a = to\text{-}V\ b \wedge b \notin total\text{-}strip\ J\rangle$
by (*metis Rep-propositional-interpretation is-interpretation-def mem-Collect-eq sign.simps*(*4*) *to-V.simps*)
then show $\langle neg\ a \notin total\text{-}strip\ J\rangle$
using *a-in* **by** (*metis neg.simps to-V.elims*)
qed

lemma *neg-in-total-strip* [*simp*]: $\langle(neg\ a \in total\text{-}strip\ J) = (a \notin total\text{-}strip\ J)\rangle$
proof
assume *neg-a-notin*: $\langle neg\ a \in total\text{-}strip\ J\rangle$
have $\langle\exists\, b.\ to\text{-}V\ a = to\text{-}V\ b \wedge b \notin total\text{-}strip\ J\rangle$
by (*metis Rep-propositional-interpretation is-interpretation-def mem-Collect-eq sign.simps*(*4*) *to-V.simps*)
then show $\langle a \notin total\text{-}strip\ J\rangle$
using *neg-a-notin* **by** (*metis neg.simps to-V.elims*)
next
assume *a-in*: $\langle a \notin total\text{-}strip\ J\rangle$
then have $\langle\exists\, b.\ to\text{-}V\ a = to\text{-}V\ b \wedge b \in total\text{-}strip\ J\rangle$
by (*metis Rep-total-interpretation belong-to.rep-eq mem-Collect-eq total-def*)
then show $\langle neg\ a \in total\text{-}strip\ J\rangle$
using *a-in* **by** (*metis neg.simps to-V.elims*)
qed

setup-lifting *type-definition-total-interpretation*

lift-definition *belong-to-total* :: *'v sign* ⇒ *'v total-interpretation* ⇒ *bool* (**infix** $\in_t$

90)
is $(\in_J)$::('*v sign* ⇒ '*v propositional-interpretation* ⇒ *bool*) **.**

lemma *in-total-to-strip* [*simp*]: ‹*a* $\in_t$ *J* ⟷ *a* ∈ *total-strip J*›
unfolding *belong-to-total-def belong-to-def* **by** *simp*

lemma *neg-prop-interp*: ‹(*v*::'*v sign*) $\in_J$ *J* ⟹ ¬ ((*neg v*) $\in_J$ *J*)›
proof *transfer*
fix *v J*
assume
j-is: ‹*is-interpretation* (*J*:: '*v sign set*)› **and**
v-in: ‹*v* ∈ *J*›
then show ‹¬ (*neg v*) ∈ *J*›
proof (*cases v*)
case (*Pos C*)
then show *?thesis*
using *is-Pos.simps*
by (*metis is-interpretation-def j-is neg.simps*(*1*) *to-V.simps v-in*)
next
case (*Neg C*)
then show *?thesis*
using *j-is v-in*
using *is-interpretation-def* **by** *fastforce*
qed
qed

lemma *neg-total-interp*: ‹(*v*::'*v sign*) $\in_t$ *J* ⟹ ¬ ((*neg v*) $\in_t$ *J*)›
proof *transfer*
fix *v J*
assume *v-in*: ‹*v* $\in_J$ (*J* :: '*v propositional-interpretation*)›
show ‹¬ *neg v* $\in_J$ *J*›
using *neg-prop-interp*[*OF v-in*] **by** *simp*
qed

lemma *neg-notin-total-interp*: ‹¬ (*v* $\in_t$ *J*) ⟹ ((*neg v*) $\in_t$ *J*)›
proof *transfer*
fix *v*::'*v sign* **and** *J*
assume
tot: ‹*total J*› **and**
not-in: ‹¬ *v* $\in_J$ *J*›
show ‹*neg v* $\in_J$ *J*›
using *tot not-in* **unfolding** *total-def*
by (*metis belong-to-total.abs-eq eq-onp-def in-total-to-strip neg-in-total-strip tot*)
qed

definition *to-AF* :: '*f* ⇒ ('*f*, '*v*::*countable*) *AF* **where**
‹*to-AF C* = *Pair C* {||}›

lemma *F-of-to-AF*: ‹*F-of* (*to-AF* $\mathcal{C}$) = $\mathcal{C}$›

unfolding *to-AF-def*
by *auto*

lemma *A-of-to-AF*: ‹*A-of* (*to-AF* $\mathcal{C}$) = {||}›
unfolding *to-AF-def*
by *auto*

lemma *F-of-circ-to-AF-is-id* [*simp*]: ‹*F-of* ∘ *to-AF* = *id*›
by (*fastforce simp*: *F-of-to-AF*)

lemma *A-of-circ-to-AF-is-empty-set* [*simp*]: ‹*A-of* ∘ *to-AF* = (λ -. {||})›
by (*fastforce simp*: *A-of-to-AF*)

lemma *F-of-propositional-clauses* [*simp*]:
‹(∀ *x* ∈ *set* $\mathcal{N}$. *F-of x* = *bot*) ⟹ *map F-of* $\mathcal{N}$ = *map* (λ -. *bot*) $\mathcal{N}$›
using *map-eq-conv*
by *blast*

lemma *F-of-Pair* [*simp*]: ‹*F-of* ∘ (λ(*x*, *y*). *AF.Pair x y*) = (λ(*x*, *y*). *x*)›
by (*smt* (*verit*, *ccfv-SIG*) *AF.sel*(*1*) *comp-apply cond-case-prod-eta old.prod.case*)

lemma *A-of-Pair* [*simp*]: ‹*A-of* ∘ (λ(*x*, *y*). *AF.Pair x y*) = (λ(*x*, *y*). *y*)›
by *fastforce*

lemma *map-A-of-map2-Pair*: ‹*length A* = *length B* ⟹ *map A-of* (*map2 AF.Pair A B*) = *B*›
proof −
assume ‹*length A* = *length B*›
then have ‹*map2* (λ *x y*. *y*) *A B* = *B*›
by (*metis map-eq-conv map-snd-zip snd-def*)
then show *?thesis*
by *auto*
qed

definition *Neg-set* :: *'v sign set* ⇒ *'v sign set* (∼- *55*) **where**
‹∼*V* ≡ {*neg v* |*v*. *v* ∈ *V*}›

definition *F-of-Inf* :: ((*'f*, *'v*::*countable*) *AF*) *inference* ⇒ *'f inference* **where**
‹*F-of-Inf* ι_{AF} = (*Infer* (*map F-of* (*prems-of* ι_{AF})) (*F-of* (*concl-of* ι_{AF})))›

6 Lifting Calculi to Add Annotations

locale *calculus-with-annotated-consrel* = *sound-calculus bot Inf entails entails-sound* Red_I Red_F
for
bot :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨ *50*) **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨*s* *50*) **and**

$Red_I :: 'f\ set \Rightarrow 'f\ inference\ set$ **and**
$Red_F :: 'f\ set \Rightarrow 'f\ set$
+ **fixes**
$fml :: \langle 'v :: countable \Rightarrow 'f\rangle$ **and**
$asn :: \langle 'f\ sign \Rightarrow 'v\ sign\ set\rangle$
assumes
fml-entails-C: $\langle\forall\ a \in asn\ C.\ sound\text{-}cons.entails\text{-}neg\ \{map\text{-}sign\ fml\ a\}\ \{C\}\rangle$
and
C-entails-fml: $\langle\forall\ a \in asn\ C.\ sound\text{-}cons.entails\text{-}neg\ \{C\}\ \{map\text{-}sign\ fml\ a\}\rangle$
and
asn-not-empty: $\langle asn\ C \neq \{\}\rangle$
begin

notation *sound-cons.entails-neg* (**infix** $\langle\models s_\sim\rangle$ *50*)

lemma *equi-entails-if-a-in-asns*: $\langle a \in asn\ C \Longrightarrow a \in asn\ D \Longrightarrow \{C\} \models s_\sim \{D\} \wedge \{D\} \models s_\sim \{C\}\rangle$
by (*smt* (*verit*) *C-entails-fml Un-commute consequence-relation.entails-cut fml-entails-C sound-cons.ext-cons-rel sup-bot-left*)

lemma *equi-entails-if-neg-a-in-asn*:
$\langle a \in asn\ C \Longrightarrow neg\ a \in asn\ D \Longrightarrow \{C\} \models s_\sim \{neg\ D\} \wedge \{neg\ D\} \models s_\sim \{C\}\rangle$
proof (*intro conjI*)
assume *a-in-asn-C*: $\langle a \in asn\ C\rangle$ **and**
neg-a-in-asn-D: $\langle neg\ a \in asn\ D\rangle$

have *fml-neg-is-neg-fml*: $\langle map\text{-}sign\ fml\ (neg\ x) = neg\ (map\text{-}sign\ fml\ x)\rangle$ **for** *x*
by (*smt* (*verit*, *ccfv-threshold*) *neg.simps*(*1*) *neg-neg-A-is-A sign.simps*(*10*) *sign.simps*(*9*) *to-V.elims*)

have $\langle\{C\} \models s_\sim \{map\text{-}sign\ fml\ a\}\rangle$
using *a-in-asn-C C-entails-fml*
by *blast*
then have $\langle\{C\} \models s_\sim \{neg\ D,\ map\text{-}sign\ fml\ a\}\rangle$
by (*smt* (*verit*, *best*) *Un-upper2 consequence-relation.entails-subsets insert-is-Un sound-cons.ext-cons-rel sup-ge1*)
moreover have $\langle\{D\} \models s_\sim \{map\text{-}sign\ fml\ (neg\ a)\}\rangle$
using *neg-a-in-asn-D C-entails-fml*
by *blast*
then have $\langle\{neg\ (neg\ D)\} \models s_\sim \{neg\ (map\text{-}sign\ fml\ a)\}\rangle$
by (*simp add*: *fml-neg-is-neg-fml*)
then have $\langle\{map\text{-}sign\ fml\ a\} \models s_\sim \{neg\ D\}\rangle$
using *sound-cons.swap-neg-in-entails-neg*
by *blast*
then have $\langle\{map\text{-}sign\ fml\ a,\ C\} \models s_\sim \{neg\ D\}\rangle$
by (*smt* (*verit*, *best*) *consequence-relation.entails-subsets insert-is-Un sound-cons.ext-cons-rel sup-ge1*)
ultimately show $\langle\{C\} \models s_\sim \{neg\ D\}\rangle$

using *consequence-relation.entails-cut*
by (*smt* (*verit*, *ccfv-threshold*) *Un-commute insert-is-Un sound-cons.ext-cons-rel sup.idem*)
next
assume *a-in-asn-C*: ‹$a \in asn\ C$› **and**
neg-a-in-asn-D: ‹$neg\ a \in asn\ D$›

have *fml-neg-is-neg-fml*: ‹$map\text{-}sign\ fml\ (neg\ x) = neg\ (map\text{-}sign\ fml\ x)$› **for** x
by (*smt* (*verit*, *ccfv-threshold*) *neg.simps*(1) *neg-neg-A-is-A sign.simps*(10) *sign.simps*(9) *to-V.elims*)

have ‹$\{map\text{-}sign\ fml\ a\} \models s_\sim \{C\}$›
using *a-in-asn-C fml-entails-C*
by *blast*
then have ‹$\{neg\ D, map\text{-}sign\ fml\ a\} \models s_\sim \{C\}$›
by (*smt* (*verit*, *best*) *Un-upper2 consequence-relation.entails-subsets insert-is-Un sound-cons.ext-cons-rel sup-ge1*)
moreover have ‹$\{map\text{-}sign\ fml\ (neg\ a)\} \models s_\sim \{D\}$›
using *neg-a-in-asn-D fml-entails-C*
by *blast*
then have ‹$\{neg\ (map\text{-}sign\ fml\ a)\} \models s_\sim \{neg\ (neg\ D)\}$›
by (*simp add*: *fml-neg-is-neg-fml*)
then have ‹$\{neg\ D\} \models s_\sim \{map\text{-}sign\ fml\ a\}$›
using *sound-cons.swap-neg-in-entails-neg*
by *blast*
then have ‹$\{neg\ D\} \models s_\sim \{map\text{-}sign\ fml\ a, C\}$›
by (*smt* (*verit*, *best*) *consequence-relation.entails-subsets insert-is-Un sound-cons.ext-cons-rel sup-ge1*)
ultimately show ‹$\{neg\ D\} \models s_\sim \{C\}$›
using *consequence-relation.entails-cut*
by (*smt* (*verit*, *ccfv-threshold*) *Un-commute insert-is-Un sound-cons.ext-cons-rel sup.idem*)
qed

definition *ιF-of* :: $('f, 'v)\ AF\ inference \Rightarrow 'f\ inference$ **where**
‹$\iota F\text{-}of\ \iota = Infer\ (List.map\ F\text{-}of\ (prems\text{-}of\ \iota))\ (F\text{-}of\ (concl\text{-}of\ \iota))$›

definition *propositional-projection* :: $('f, 'v)\ AF\ set \Rightarrow ('f, 'v)\ AF\ set\ (proj_\bot)$ **where**
‹$proj_\bot\ \mathcal{N} = \{\mathcal{C}.\ \mathcal{C} \in \mathcal{N} \wedge F\text{-}of\ \mathcal{C} = bot\}$›

lemma *prop-proj-in*: ‹$proj_\bot\ \mathcal{N} \subseteq \mathcal{N}$›
unfolding *propositional-projection-def* **by** *blast*

definition *enabled* :: $('f, 'v)\ AF \Rightarrow 'v\ total\text{-}interpretation \Rightarrow bool$ **where**
$enabled\ \mathcal{C}\ J \equiv fset\ (A\text{-}of\ \mathcal{C}) \subseteq (total\text{-}strip\ J)$

lemma *subformula-of-enabled-formula-is-enabled*: ‹$A\text{-}of\ \mathcal{C}\ |\subset|\ A\text{-}of\ \mathcal{C}' \Longrightarrow enabled$

$\mathcal{C}'\ J \Longrightarrow$ *enabled* $\mathcal{C}\ J$›
unfolding *enabled-def*
by (*meson less-eq-fset.rep-eq pfsubset-imp-fsubset subset-trans*)

lemma *enabled-iff*: ‹*A-of* $\mathcal{C}$ = *A-of* $\mathcal{C}' \Longrightarrow$ *enabled* $\mathcal{C}\ J \longleftrightarrow$ *enabled* $\mathcal{C}'\ J$›
unfolding *enabled-def*
by *simp*

definition *enabled-set* :: ('*f*, '*v*) *AF set* ⇒ '*v total-interpretation* ⇒ *bool* **where**
‹*enabled-set* $\mathcal{N}\ J = (\forall \mathcal{C} \in \mathcal{N}.$ *enabled* $\mathcal{C}\ J)$›

lemma *enabled-set-singleton* [*simp*]: ‹*enabled-set* $\{\mathcal{C}\}\ J \longleftrightarrow$ *enabled* $\mathcal{C}\ J$›
by (*auto simp add*: *enabled-set-def*)

definition *enabled-inf* :: ('*f*, '*v*) *AF inference* ⇒ '*v total-interpretation* ⇒ *bool* **where**
‹*enabled-inf* $\iota\ J = (\forall \mathcal{C} \in$ *set* (*prems-of* ι). *enabled* $\mathcal{C}\ J)$›

definition *enabled-projection* :: ('*f*, '*v*) *AF set* ⇒ '*v total-interpretation* ⇒ '*f set*
(**infix** $proj_J$ *60*) **where**
‹$\mathcal{N}\ proj_J\ J = \{$*F-of* $\mathcal{C}\ |\mathcal{C}.\ \mathcal{C} \in \mathcal{N} \wedge$ *enabled* $\mathcal{C}\ J\}$›

lemma *Union-of-enabled-projection-is-enabled-projection*: ‹$(\bigcup\ \mathcal{C} \in \mathcal{N}.\ \{\mathcal{C}\}\ proj_J\ \mathcal{J}) = \mathcal{N}\ proj_J\ \mathcal{J}$›
unfolding *enabled-projection-def*
by *blast*

lemma *projection-of-enabled-subset*:
‹*fset* $B \subseteq$ *total-strip* $J \Longrightarrow \{$*AF.Pair* $C\ (A\ |\cup|\ B)\}\ proj_J\ J = \{$*AF.Pair* $C\ A\}\ proj_J\ J$›
unfolding *enabled-projection-def enabled-def*
by *auto*

lemma *Un-of-enabled-projection-is-enabled-projection-of-Un*:
‹$(\bigcup\ x.\ P\ x)\ proj_J\ J = (\bigcup\ x.\ P\ x\ proj_J\ J)$›
proof (*intro subset-antisym subsetI*)
fix *x*
assume ‹$x \in (\bigcup\ x.\ P\ x)\ proj_J\ J$›
then have ‹$\exists\ y.\ x =$ *F-of* $y \wedge$ *enabled* $y\ J \wedge y \in (\bigcup\ x.\ P\ x)$›
unfolding *enabled-projection-def*
by *blast*
then have ‹$\exists\ y.\ x =$ *F-of* $y \wedge$ *enabled* $y\ J \wedge (\exists\ z.\ y \in P\ z)$›
by *blast*
then have ‹$\exists\ y.\ \exists\ z.\ x =$ *F-of* $y \wedge$ *enabled* $y\ J \wedge y \in P\ z$›
by *blast*
then show ‹$x \in (\bigcup\ x.\ P\ x\ proj_J\ J)$›
unfolding *enabled-projection-def*
by *blast*
next

fix x
assume ‹$x \in (\bigcup\ x.\ P\ x\ proj_J\ J)$›
then have ‹$\exists\ y.\ x \in P\ y\ proj_J\ J$›
by *blast*
then have ‹$\exists\ y.\ \exists\ z.\ x = F\text{-}of\ z \wedge enabled\ z\ J \wedge z \in P\ y$›
unfolding *enabled-projection-def*
by *blast*
then show ‹$x \in (\bigcup\ x.\ P\ x)\ proj_J\ J$›
unfolding *enabled-projection-def*
by *blast*
qed

lemma *enabled-projection-of-Int-is-Int-of-enabled-projection*:
‹$x \in (\bigcap\ S)\ proj_J\ J \Longrightarrow x \in \bigcap\ \{\ x\ proj_J\ J \mid x.\ x \in S\ \}$›
unfolding *enabled-projection-def*
by *blast*

definition *enabled-projection-Inf* :: ('*f*, '*v*) *AF inference set* ⇒ '*v total-interpretation* ⇒
'*f inference set* (**infix** $\iota proj_J$ *60*) **where**
‹$I\ \iota proj_J\ J = \{\iota F\text{-}of\ \iota \mid \iota.\ \iota \in I \wedge enabled\text{-}inf\ \iota\ J\}$›

fun *fml-ext* :: '*v sign* ⇒ '*f sign* **where**
fml-ext (*Pos v*) = *Pos* (*fml v*) |
fml-ext (*Neg v*) = *Neg* (*fml v*)

lemma *fml-ext-is-mapping*: ‹*fml-ext v* = *map-sign fml v*›
by (*metis fml-ext.cases fml-ext.simps*(*1*) *fml-ext.simps*(*2*) *sign.simps*(*10*) *sign.simps*(*9*))

lemma *fml-ext-preserves-sign*: *is-Pos v* ≡ *is-Pos* (*fml-ext v*)
by (*induct v*, *auto*)

lemma *to-V-fml-ext* [*simp*]: ‹*to-V* (*fml-ext v*) = *fml* (*to-V v*)›
by (*induct v*, *auto*)

lemma *fml-ext-preserves-val*: ‹*to-V v1* = *to-V v2* ⟹ *to-V* (*fml-ext v1*) = *to-V* (*fml-ext v2*)›
by *simp*

definition *sound-consistent* :: '*v total-interpretation* ⇒ *bool* **where**
‹*sound-consistent J* ≡ ¬ (*sound-cons.entails-neg* (*fml-ext* ‘ (*total-strip J*)) {*Pos bot*})›

definition *propositional-model* :: '*v total-interpretation* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*
(**infix** $\models_p$ *50*) **where**
‹$J \models_p \mathcal{N} \equiv bot \notin ((proj_\bot\ \mathcal{N})\ proj_J\ J)$›

lemma ‹$J \models_p \{\}$›

unfolding *propositional-model-def enabled-projection-def propositional-projection-def*
by *blast*

The definition below is essentially the same as the one above since term($proj_\perp\ \mathcal{N}$) $proj_J\ J$ is either empty or contains only bot

definition *propositional-model2* :: *'v total-interpretation* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool*
(**infix** $\models_p 2$ *50*) **where**
‹$J \models_p 2\ \mathcal{N} \equiv (\{\} = ((proj_\perp\ \mathcal{N})\ proj_J\ J))$›

lemma *subset-model-p2*: ‹$\mathcal{N}' \subseteq \mathcal{N} \Longrightarrow J \models_p 2\ \mathcal{N} \Longrightarrow J \models_p 2\ \mathcal{N}'$›
by (*simp add*: *enabled-projection-def propositional-model2-def propositional-projection-def*

subset-eq)

lemma *subset-not-model*: ‹$\neg\ J \models_p 2\ \mathcal{N} \Longrightarrow \mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2 \Longrightarrow J \models_p 2\ \mathcal{N}_1 \Longrightarrow \neg\ J \models_p 2\ \mathcal{N}_2$›
unfolding *propositional-model2-def propositional-projection-def enabled-projection-def*
by *blast*

lemma *supset-not-model-p2*: ‹$\mathcal{N}' \subseteq \mathcal{N} \Longrightarrow \neg\ J \models_p 2\ \mathcal{N}' \Longrightarrow \neg\ J \models_p 2\ \mathcal{N}$›
using *subset-model-p2* **by** *blast*

fun *sign-to-atomic-formula* :: *'v sign* ⇒ *'v formula* **where**
‹*sign-to-atomic-formula* (*Pos v*) = *Atom v*› |
‹*sign-to-atomic-formula* (*Neg v*) = *Not* (*Atom v*)›

definition *sign-set-to-formula-set* :: *'v sign set* ⇒ *'v formula set* **where**
‹*sign-set-to-formula-set A* = *sign-to-atomic-formula* ‘ *A*›

lemma *form-shape-sign-set*: ‹∀ *f*∈*sign-set-to-formula-set A*. ∃ *v*. *f* = *Atom v* ∨ *f* = *Not* (*Atom v*)›
unfolding *sign-set-to-formula-set-def*
by (*metis image-iff sign-to-atomic-formula.elims*)

definition *AF-to-formula-set* :: (*'f*, *'v*) *AF* ⇒ *'v formula set* **where**

‹*AF-to-formula-set* $\mathcal{C}$ = *sign-set-to-formula-set* (*neg* ‘ *fset* (*A-of* $\mathcal{C}$))›

definition *AF-to-formula* :: (*'f*, *'v*) *AF* ⇒ *'v formula* **where**
‹*AF-to-formula* $\mathcal{C}$ = *BigOr* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$))))›

lemma *form-shape-AF*: ‹∀ *f*∈*AF-to-formula-set* $\mathcal{C}$. ∃ *v*. *f* = *Atom v* ∨ *f* = *Not* (*Atom v*)›
using *form-shape-sign-set* **unfolding** *AF-to-formula-set-def* **by** *simp*

definition *AF-proj-to-formula-set-set* :: (*'f*, *'v*) *AF set* ⇒ *'v formula set set* **where**

‹*AF-proj-to-formula-set-set* $\mathcal{N}$ = *AF-to-formula-set* ‘ ($proj_\perp\ \mathcal{N}$)›

definition *AF-proj-to-formula-set* :: $('f, 'v)$ *AF set* $\Rightarrow$ $'v$ *formula set* **where**
‹*AF-proj-to-formula-set* $\mathcal{N}$ = *AF-to-formula* ‘ ($proj_\perp$ $\mathcal{N}$)›

definition *AF-assertions-to-formula* :: $('f, 'v)$ *AF* $\Rightarrow$ $'v$ *formula* **where**
‹*AF-assertions-to-formula* $\mathcal{C}$ = *BigAnd* (*map sign-to-atomic-formula* (*list-of-fset* (*A-of* $\mathcal{C}$)))›

definition *AF-assertions-to-formula-set* :: $('f, 'v)$ *AF set* $\Rightarrow$ $'v$ *formula set* **where**
‹*AF-assertions-to-formula-set* $\mathcal{N}$ = *AF-assertions-to-formula* ‘ $\mathcal{N}$›

lemma *F-to-C-set*: ‹$\forall$ *F*$\in$*AF-proj-to-formula-set-set* $\mathcal{N}$. $\exists$ $\mathcal{C}$$\in$$proj_\perp$ $\mathcal{N}$. *F* = *sign-to-atomic-formula* ‘ *neg* ‘ *fset* (*A-of* $\mathcal{C}$)›
unfolding *AF-proj-to-formula-set-set-def AF-to-formula-set-def sign-set-to-formula-set-def*
by *auto*

lemma *F-to-C*: ‹$\forall$ *F*$\in$*AF-proj-to-formula-set* $\mathcal{N}$. $\exists$ $\mathcal{C}$$\in$$proj_\perp$ $\mathcal{N}$. *F* = *BigOr* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$))))›
unfolding *AF-proj-to-formula-set-def AF-to-formula-def* **by** *auto*

lemma *C-to-F-set*: ‹$\forall$ $\mathcal{C}$$\in$$proj_\perp$ $\mathcal{N}$. $\exists$ *F*$\in$*AF-proj-to-formula-set-set* $\mathcal{N}$. *F* = *sign-to-atomic-formula* ‘ *neg* ‘ *fset* (*A-of* $\mathcal{C}$)›
unfolding *AF-proj-to-formula-set-set-def AF-to-formula-set-def sign-set-to-formula-set-def*
by *auto*

lemma *C-to-F*: ‹$\forall$ $\mathcal{C}$$\in$$proj_\perp$ $\mathcal{N}$. $\exists$ *F*$\in$*AF-proj-to-formula-set* $\mathcal{N}$. *F* = *BigOr* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$))))›
unfolding *AF-proj-to-formula-set-def AF-to-formula-def* **by** *auto*

lemma *form-shape-proj*: ‹$\forall$ *f*$\in\bigcup$ (*AF-proj-to-formula-set-set* $\mathcal{N}$). $\exists$ *v*. *f* = *Atom v* $\vee$ *f* = *Not* (*Atom v*)›
using *form-shape-AF* **unfolding** *AF-proj-to-formula-set-set-def* **by** *simp*

definition *to-valuation* :: $'v$ *total-interpretation* $\Rightarrow$ $'v$ *valuation* **where**
‹*to-valuation J* = (λ*a*. *Pos a* $\in_t$ *J*)›

lemma *val-strip-pos*: ‹*to-valuation J a* $\equiv$ *Pos a* $\in$ *total-strip J*›
unfolding *to-valuation-def belong-to-total-def belong-to-def* **by** *simp*

lemma *val-strip-neg*: ‹($\neg$ *to-valuation J a*) = (*Neg a* $\in$ *total-strip J*)›
proof −
have ‹($\neg$ *to-valuation J a*) = ($\neg$ *Pos a* $\in$ *total-strip J*)›
using *val-strip-pos* **by** *simp*
also have ‹($\neg$ *Pos a* $\in$ *total-strip J*) = (*Neg a* $\in$ *total-strip J*)›
proof
fix *a J*
assume *not-pos*: ‹*Pos* (*a*::$'v$) $\notin$ *total-strip J*›
have ‹*is-interpretation* (*total-strip J*)›
using *Rep-propositional-interpretation* **by** *blast*

then show ‹*Neg a ∈ total-strip J*›
unfolding *is-interpretation-def* **using** *total-def not-pos*
by (*metis Rep-total-interpretation belong-to.rep-eq mem-Collect-eq to-V.elims*)
next
assume *neg*: ‹*Neg a ∈ total-strip J*›
have ‹*is-interpretation* (*total-strip J*)›
using *Rep-propositional-interpretation* **by** *blast*
then show ‹*Pos a ∉ total-strip J*›
unfolding *is-interpretation-def* **using** *neg*
by (*metis sign.distinct*(*1*) *to-V.simps*(*1*) *to-V.simps*(*2*))
qed
finally show ‹(¬ *to-valuation J a*) = (*Neg a ∈ total-strip J*)› **.**
qed

lemma *equiv-prop-entails*: ‹(*J* ⊨*p* 𝒩) ⟷ (*J* ⊨*p*2 𝒩)›
unfolding *propositional-model-def propositional-model2-def propositional-projection-def enabled-projection-def*
by *blast*

definition *propositional-model3* :: *'v total-interpretation* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool*
(**infix** ⊨*p*3 *50*) **where**
‹*J* ⊨*p*3 𝒩 ≡ (∀ *F*∈*AF-proj-to-formula-set-set* 𝒩. ∃ *f*∈*F*. *formula-semantics* (*to-valuation J*) *f*)›

lemma *equiv-prop-entail2-sema*:
‹(*J* ⊨*p*2 𝒩) ⟷ (*J* ⊨*p*3 𝒩)›
unfolding *propositional-model3-def propositional-model2-def enabled-projection-def enabled-def*
proof
assume *empty-proj*: ‹{} = {*F-of* 𝒞 |𝒞. 𝒞 ∈ *proj*⊥ 𝒩 ∧ *fset* (*A-of* 𝒞) ⊆ *total-strip J*}›
then have ‹∀ 𝒞∈*proj*⊥ 𝒩. ∃ *v*∈*fset* (*A-of* 𝒞). *neg v* ∈ *total-strip J*›
by (*smt* (*verit*, *ccfv-SIG*) *empty-iff mem-Collect-eq neg.elims subsetI val-strip-neg val-strip-pos*)
show ‹∀ *F*∈*AF-proj-to-formula-set-set* 𝒩. ∃ *f*∈*F*. *formula-semantics* (*to-valuation J*) *f*›
proof
fix *F*
assume *F-in*: ‹*F* ∈ *AF-proj-to-formula-set-set* 𝒩›
then obtain 𝒞 **where** ‹𝒞 ∈ *proj*⊥ 𝒩› **and** *F-from-𝒞*: ‹*F* = *sign-to-atomic-formula* ‘ *neg* ‘ *fset* (*A-of* 𝒞)›
using *F-to-𝒞-set* **by** *meson*
then have ‹∃ *v*∈*fset* (*A-of* 𝒞). *v* ∉ *total-strip J*›
using *empty-proj* **by** *blast*
then obtain *v* **where** *v-in*: ‹*v* ∈ *fset* (*A-of* 𝒞)› **and** *v-notin*: ‹*v* ∉ *total-strip J*› **by** *auto*
define *f* **where** ‹*f* = *sign-to-atomic-formula* (*neg v*)›
then have ‹*formula-semantics* (*to-valuation J*) *f*›

using *v-notin*
by (*smt* (*z3*) *belong-to.rep-eq belong-to-total.rep-eq formula-semantics.simps neg.elims*
sign-to-atomic-formula.simps to-valuation-def val-strip-neg)
moreover have ‹$f \in F$›
using *F-from-$\mathcal{C}$ v-in f-def* **by** *blast*
ultimately show ‹$\exists f{\in}F.$ *formula-semantics* (*to-valuation J*) *f*› **by** *auto*
qed
next
assume *F-sat*: ‹$\forall F{\in}$*AF-proj-to-formula-set-set* $\mathcal{N}.$ $\exists f{\in}F.$ *formula-semantics* (*to-valuation J*) *f*›
have ‹$\forall \mathcal{C}{\in}$*proj*$_\perp$ $\mathcal{N}.$ $\exists v{\in}$*fset* (*A-of* $\mathcal{C}$). *neg v* $\in$ *total-strip J*›
proof
fix $\mathcal{C}$
assume ‹$\mathcal{C} \in$ *proj*$_\perp$ $\mathcal{N}$›
then obtain *F* **where** ‹$F \in$ *AF-proj-to-formula-set-set* $\mathcal{N}$› **and**
F-from-$\mathcal{C}$: ‹$F =$ *sign-to-atomic-formula* ‘ *neg* ‘ *fset* (*A-of* $\mathcal{C}$)›
using *$\mathcal{C}$-to-F-set* **by** *blast*
then have ‹$\exists f{\in}F.$ *formula-semantics* (*to-valuation J*) *f*›
using *F-sat* **by** *blast*
then obtain *f* **where** *f-in*: ‹$f \in F$› **and** *sat-f*: ‹*formula-semantics* (*to-valuation J*) *f*›
by *blast*
then obtain *v* **where** *v-is*: ‹$f =$ *sign-to-atomic-formula* (*neg v*)› **and** *v-in*: ‹$v \in$ *fset* (*A-of* $\mathcal{C}$)›
using *F-from-$\mathcal{C}$* **by** *blast*
then have ‹*neg v* $\in$ *total-strip J*›
using *sat-f* **unfolding** *to-valuation-def*
by (*smt* (*z3*) *belong-to.rep-eq belong-to-total.rep-eq formula-semantics.simps sign.exhaust*
sign-to-atomic-formula.simps val-strip-neg val-strip-pos)
then show ‹$\exists v{\in}$*fset* (*A-of* $\mathcal{C}$). *neg v* $\in$ *total-strip J*›
using *v-in* **by** *auto*
qed
then show ‹$\{\} = \{$*F-of* $\mathcal{C}$ $|\mathcal{C}.$ $\mathcal{C} \in$ *proj*$_\perp$ $\mathcal{N} \wedge$ *fset* (*A-of* $\mathcal{C}$) $\subseteq$ *total-strip J*$\}$›
by (*smt* (*verit*, *ccfv-threshold*) *empty-Collect-eq is-Pos.cases neg.simps*(*1*) *neg.simps*(*2*) *subsetD*
val-strip-neg val-strip-pos)
qed

lemma *equiv-prop-entail2-sema2*:
‹($J \models_p 2$ $\mathcal{N}$) $\longleftrightarrow$ ($\forall F{\in}$*AF-proj-to-formula-set* $\mathcal{N}.$ *formula-semantics* (*to-valuation J*) *F*)›
unfolding *propositional-model2-def enabled-projection-def enabled-def*
proof
assume *empty-proj*: ‹$\{\} = \{$*F-of* $\mathcal{C}$ $|\mathcal{C}.$ $\mathcal{C} \in$ *proj*$_\perp$ $\mathcal{N} \wedge$ *fset* (*A-of* $\mathcal{C}$) $\subseteq$ *total-strip J*$\}$›
show ‹$\forall F{\in}$*AF-proj-to-formula-set* $\mathcal{N}.$ *formula-semantics* (*to-valuation J*) *F*›

proof
fix F
assume *F-in*: ‹$F \in$ *AF-proj-to-formula-set* $\mathcal{N}$›
then obtain $\mathcal{C}$ **where** ‹$\mathcal{C} \in$ *proj*$_\perp$ $\mathcal{N}$› **and**
F-from-$\mathcal{C}$: ‹F = *BigOr* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$))))›
using *F-to-*$\mathcal{C}$ **by** *meson*
then have ‹$\exists\, v{\in}$*fset* (*A-of* $\mathcal{C}$). $v \notin$ *total-strip* J›
using *empty-proj* **by** *blast*
then obtain v **where** *v-in*: ‹$v \in$ *fset* (*A-of* $\mathcal{C}$)› **and** *v-notin*: ‹$v \notin$ *total-strip* J› **by** *auto*
define f **where** ‹f = *sign-to-atomic-formula* (*neg* v)›
then have ‹*formula-semantics* (*to-valuation* J) f›
using *v-notin*
by (*smt* (*z3*) *belong-to.rep-eq belong-to-total.rep-eq formula-semantics.simps neg.elims*
sign-to-atomic-formula.simps to-valuation-def val-strip-neg)
moreover have ‹$f \in$ *set* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$))))›
unfolding *f-def* **using** *fset-map2*[*OF v-in*, *of sign-to-atomic-formula neg*] **.**
ultimately have ‹$\exists f{\in}$*set* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$)))).
formula-semantics (*to-valuation* J) f›
by *blast*
then show ‹*formula-semantics* (*to-valuation* J) F›
using *BigOr-semantics*[*of to-valuation* J
map sign-to-atomic-formula (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$)))] *F-from-*$\mathcal{C}$
by *meson*
qed
next
assume *F-sat*: ‹$\forall\, F{\in}$*AF-proj-to-formula-set* $\mathcal{N}$. *formula-semantics* (*to-valuation* J) F›
have ‹$\forall \mathcal{C}{\in}$*proj*$_\perp$ $\mathcal{N}$. $\exists\, v{\in}$*fset* (*A-of* $\mathcal{C}$). *neg* $v \in$ *total-strip* J›
proof
fix $\mathcal{C}$
assume $\mathcal{C}$*-in*: ‹$\mathcal{C} \in$ *proj*$_\perp$ $\mathcal{N}$›
define F **where** ‹F = *AF-to-formula* $\mathcal{C}$›
then have ‹$F \in$ *AF-proj-to-formula-set* $\mathcal{N}$›
unfolding *AF-proj-to-formula-set-def* **using** $\mathcal{C}$*-in* **by** *blast*
then have ‹*formula-semantics* (*to-valuation* J) F› **using** *F-sat* **by** *blast*
then have ‹$\exists f{\in}$*set* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$)))).
formula-semantics (*to-valuation* J) f›
using *BigOr-semantics*[*of to-valuation* J] **unfolding** *F-def AF-to-formula-def* **by** *simp*
then obtain f **where**
f-in: ‹$f \in$ *set* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset* (*A-of* $\mathcal{C}$))))›
and *val-f*: ‹*formula-semantics* (*to-valuation* J) f› **by** *blast*
obtain v **where** *v-in*: ‹$v \in$ *fset* (*A-of* $\mathcal{C}$)› **and** *f-is*: ‹f = *sign-to-atomic-formula*

(neg v)›
using *f-in* **unfolding** *list-of-fset-def*
by (*smt* (*z3*) *exists-fset-of-list fset-of-list.rep-eq image-iff list.set-map someI*)
have ‹*neg v* ∈ *total-strip J*›
using *f-is val-f* **unfolding** *to-valuation-def*
by (*metis* (*mono-tags*, *lifting*) *belong-to.rep-eq belong-to-total.rep-eq*
formula-semantics.simps(*1*) *formula-semantics.simps*(*3*) *sign-to-atomic-formula.cases*
sign-to-atomic-formula.simps(*1*) *sign-to-atomic-formula.simps*(*2*) *val-f*
val-strip-neg)
then show ‹∃ *v*∈*fset* (*A-of* C). *neg v* ∈ *total-strip J*›
using *v-in* **by** *blast*
qed
then show ‹{} = {*F-of* C |C. C ∈ *proj*⊥ N ∧ *fset* (*A-of* C) ⊆ *total-strip J*}›
by (*smt* (*verit*, *ccfv-threshold*) *empty-Collect-eq is-Pos.cases neg.simps*(*1*) *neg.simps*(*2*)
subsetD
val-strip-neg val-strip-pos)
qed

lemma *equiv-prop-entail-sema*:
‹(*J* |=*p* N) ⟷ (*J* |=*p*3 N)›
using *equiv-prop-entails equiv-prop-entail2-sema* **by** *presburger*

lemma ‹*f* ‘ *fset A* = *set* (*map f* (*list-of-fset A*))›
proof
show ‹*f* ‘ *fset A* ⊆ *set* (*map f* (*list-of-fset A*))›
proof
fix *v*
assume *v-in*: ‹*v* ∈ *f* ‘ *fset A*›
then obtain *a* **where** *a* |∈| *A f a* = *v*
by *blast*
then show ‹*v* ∈ *set* (*map f* (*list-of-fset A*))›
unfolding *list-of-fset-def*
by (*metis fset-map2 list-of-fset-def map-ident*)
qed
next
show ‹*set* (*map f* (*list-of-fset A*)) ⊆ *f* ‘ *fset A*›
proof
fix *v*
assume ‹*v* ∈ *set* (*map f* (*list-of-fset A*))›
then obtain *a* **where** *a* |∈| *A f a* = *v*
unfolding *list-of-fset-def*
by (*metis* (*mono-tags*, *lifting*) *exists-fset-of-list fimage.rep-eq fimageE fset-of-list.rep-eq*
set-map someI-ex)
then show ‹*v* ∈ *f* ‘ *fset A*›
by *blast*
qed
qed

lemma *equiv-prop-sema1-sema2*:
‹$(J \models_p 3\ \mathcal{N}) \longleftrightarrow$
$(\forall F{\in}AF\text{-}proj\text{-}to\text{-}formula\text{-}set\ \mathcal{N}.\ formula\text{-}semantics\ (to\text{-}valuation\ J)\ F)$›
using *equiv-prop-entail2-sema2 equiv-prop-entail2-sema*
unfolding *propositional-model3-def* **by** *auto*

lemma *equiv-enabled-assertions-sema*:
‹$(enabled\text{-}set\ \mathcal{N}\ J) \longleftrightarrow (\forall F{\in}AF\text{-}assertions\text{-}to\text{-}formula\text{-}set\ \mathcal{N}.\ formula\text{-}semantics\ (to\text{-}valuation\ J)\ F)$›
unfolding *enabled-projection-def enabled-def enabled-set-def*
proof
assume *enab-N*: ‹$\forall \mathcal{C}{\in}\mathcal{N}.\ fset\ (A\text{-}of\ \mathcal{C}) \subseteq total\text{-}strip\ J$›
show ‹$\forall F{\in}AF\text{-}assertions\text{-}to\text{-}formula\text{-}set\ \mathcal{N}.\ formula\text{-}semantics\ (to\text{-}valuation\ J)\ F$›
proof
fix F
assume *F-in*: ‹$F \in AF\text{-}assertions\text{-}to\text{-}formula\text{-}set\ \mathcal{N}$›
then obtain $\mathcal{C}$ **where** $\mathcal{C}$*-in*: ‹$\mathcal{C} \in \mathcal{N}$› **and**
F-from-$\mathcal{C}$: ‹$F = BigAnd\ (map\ sign\text{-}to\text{-}atomic\text{-}formula\ (list\text{-}of\text{-}fset\ (A\text{-}of\ \mathcal{C})))$›
unfolding *AF-assertions-to-formula-def AF-assertions-to-formula-set-def* **by** *auto*
have ‹$\forall f{\in}set\ (map\ sign\text{-}to\text{-}atomic\text{-}formula\ (list\text{-}of\text{-}fset\ (A\text{-}of\ \mathcal{C}))).$
$formula\text{-}semantics\ (to\text{-}valuation\ J)\ f$›
proof
fix f
assume *f-in*: ‹$f \in set\ (map\ sign\text{-}to\text{-}atomic\text{-}formula\ (list\text{-}of\text{-}fset\ (A\text{-}of\ \mathcal{C})))$›
define L **where** ‹$L = (list\text{-}of\text{-}fset\ (A\text{-}of\ \mathcal{C}))$›
then obtain $a\ v$ **where** *f-is*: ‹$sign\text{-}to\text{-}atomic\text{-}formula\ a = f$› **and** *a-in*: ‹$a \in set\ L$›
and *v-is*: ‹$to\text{-}V\ a = v$›
using *f-in* **by** *auto*
have ‹$a \in fset\ (A\text{-}of\ \mathcal{C})$›
using *a-in* **unfolding** *L-def* **by** (*smt* (*verit*, *ccfv-threshold*) *exists-fset-of-list fset-of-list.rep-eq list-of-fset-def someI-ex*)
then have *a-in*: ‹$a \in total\text{-}strip\ J$›
using *enab-N* $\mathcal{C}$*-in* **by** *blast*
consider (*Pos*) $a = Pos\ v$ | (*Neg*) ‹$a = Neg\ v$›
using *v-is is-Neg-to-V is-Pos-to-V* **by** *blast*
then show ‹$formula\text{-}semantics\ (to\text{-}valuation\ J)\ f$›
proof *cases*
case *Pos*
then have ‹$f = Atom\ v$› **using** *f-is* **by** *auto*
then show *?thesis*
using *a-in* **unfolding** *to-valuation-def belong-to-total-def belong-to-def*
by (*simp add*: *Pos*)
next
case *Neg*

```
      then have ‹f = Not (Atom v)› using f-is by auto
      then show ?thesis
        using a-in Neg to-valuation-def val-strip-neg by force
    qed
  qed
  then show ‹formula-semantics (to-valuation J) F›
    using BigAnd-semantics[of to-valuation J
        map sign-to-atomic-formula (list-of-fset (A-of C))] F-from-C by blast
 qed
next
 assume F-sat: ‹∀ F∈AF-assertions-to-formula-set 𝒩. formula-semantics (to-valuation
J) F›
  have ‹∀ C∈𝒩. ∀ a∈fset (A-of C). a ∈ total-strip J›
  proof clarify
    fix C a
    assume
      C-in: ‹C ∈ 𝒩› and
      a-in: ‹a ∈ fset (A-of C)›
    define F where ‹F = AF-assertions-to-formula C›
    then have ‹F ∈ AF-assertions-to-formula-set 𝒩›
      unfolding AF-assertions-to-formula-set-def using C-in by blast
    then have ‹formula-semantics (to-valuation J) F› using F-sat by blast
     then have all-f-sat: ‹∀ f∈set (map sign-to-atomic-formula (list-of-fset (A-of
C))).
      formula-semantics (to-valuation J) f›
     using BigAnd-semantics[of to-valuation J] unfolding F-def AF-assertions-to-formula-def
      by simp
    define f where ‹f = sign-to-atomic-formula a›
    then have ‹f ∈ set (map sign-to-atomic-formula (list-of-fset (A-of C)))›
      using a-in fset-map2 by fastforce
    then have f-sat: ‹formula-semantics (to-valuation J) f›
      using all-f-sat by auto
    then show ‹a ∈ total-strip J›
      using f-def unfolding to-valuation-def
     by (metis f-sat belong-to.rep-eq belong-to-total.rep-eq formula-semantics.simps(1)
         formula-semantics.simps(3) sign-to-atomic-formula.elims val-strip-neg)
  qed
  then show ‹∀ C∈𝒩. fset (A-of C) ⊆ total-strip J› by blast
qed

definition sound-propositional-model :: 'v total-interpretation ⇒ ('f, 'v) AF set
⇒ bool
  (infix ⊨s_p 50) where
  ‹J ⊨s_p 𝒩 ≡ (bot ∉ ((enabled-projection (propositional-projection 𝒩) J)) ∨
    ¬ sound-consistent J)›

definition propositionally-unsatisfiable :: ('f, 'v) AF set ⇒ bool where
  ‹propositionally-unsatisfiable 𝒩 ≡ ∀ J. ¬ (J ⊨_p 𝒩)›
```

lemma *unsat-simp*:
assumes
‹¬ *sat* (*S′* ∪ *S*::*′v formula set*)›
‹*sat S′*›
‹⋃ (*atoms* ‘ *S′*) ∩ ⋃ (*atoms* ‘ *S*) = {}›
shows
‹¬ *sat S*›
unfolding *sat-def*
proof
assume *contra*: ‹∃ *A*. ∀ *F*∈*S*. *formula-semantics A F*›
then obtain *AS* **where** *AS-is*: ‹∀ *F*∈*S*. *formula-semantics AS F*› **by** *blast*
obtain *AF* **where** *AF-is*: ‹∀ *F*∈*S′*. *formula-semantics AF F*› **using** *assms*(*2*)
unfolding *sat-def* **by** *blast*
define *A* **where** ‹*A* = (λ*a*. *if a* ∈ ⋃ (*atoms* ‘ *S′*) *then AF a else AS a*)›
have ‹∀ *F*∈*S′*. *formula-semantics A F*›
using *AF-is relevant-atoms-same-semantics* **unfolding** *A-def*
by (*smt* (*verit*, *best*) *UN-I*)
moreover have ‹∀ *F*∈*S*. *formula-semantics A F*›
using *AS-is relevant-atoms-same-semantics assms*(*3*) **unfolding** *A-def*
by (*smt* (*verit*, *del-insts*) *Int-iff UN-I empty-iff*)
ultimately have ‹∀ *F′*∈(*S′*∪*S*). *formula-semantics A F′*› **by** *blast*
then show *False*
using *assms*(*1*) **unfolding** *sat-def* **by** *blast*
qed

lemma *proj-to-form-un*: ‹*AF-proj-to-formula-set* (*A* ∪ *B*) =
AF-proj-to-formula-set A ∪ *AF-proj-to-formula-set B*›
unfolding *AF-proj-to-formula-set-def propositional-projection-def* **by** *blast*

lemma *unsat-AF-simp*:
assumes
‹¬ *sat* (*AF-proj-to-formula-set* (*S′* ∪ *S*))›
‹*sat* (*AF-proj-to-formula-set S′*)›
‹⋃ (*atoms* ‘ (*AF-proj-to-formula-set S′*)) ∩ ⋃ (*atoms* ‘ (*AF-proj-to-formula-set*
S)) = {}›
shows
‹¬ *sat* (*AF-proj-to-formula-set S*)›
using *unsat-simp assms proj-to-form-un* **by** *metis*

lemma *set-list-of-fset*[*simp*]: ‹*set* (*list-of-fset A*) = *fset A*›
unfolding *list-of-fset-def*
by (*smt* (*verit*, *del-insts*) *exists-fset-of-list fset-of-list.rep-eq someI-ex*)

lemma *vars-in-assertion*: ‹*to-V* ‘ (*set* (*list-of-fset A*)) = *to-V* ‘ (*fset A*)›
by *simp*

lemma *atoms-bigor*: ‹*atoms* (*BigOr L*) = ⋃ (*atoms* ‘ (*set L*))›
unfolding *BigOr-def* **by** (*induction L*) *auto*

lemma *atoms-neg*: ‹*atoms* (*sign-to-atomic-formula* (*neg A*)) = *atoms* (*sign-to-atomic-formula A*)›
by (*metis formula.simps*(*92*) *neg.elims sign-to-atomic-formula.simps*(*1*) *sign-to-atomic-formula.simps*(*2*))

lemma *set-maps-list-of-fset*: ‹*set* (*map sign-to-atomic-formula* (*map neg* (*list-of-fset A*))) =
sign-to-atomic-formula ‘ *neg* ‘ *fset A*›
using *set-map* **by** *auto*

lemma *atoms-to-V-mono*: ‹*atoms* (*sign-to-atomic-formula A*) = {*to-V A*}›
by (*metis formula.set*(*1*) *formula.set*(*3*) *sign-to-atomic-formula.simps*(*1*) *sign-to-atomic-formula.simps*(*2*) *to-V.elims*)

lemma *atoms-to-V*: ‹⋃(*atoms* ‘ *sign-to-atomic-formula* ‘ *A*) = *to-V* ‘ *A*›
proof −
have ‹⋃(*atoms* ‘ *sign-to-atomic-formula* ‘ *A*) = ⋃{{*to-V a*} |*a*. *a* ∈ *A*}›
using *atoms-to-V-mono* **by** *auto*
also have ‹... = *to-V* ‘ *A*›
by *blast*
finally show ‹⋃(*atoms* ‘ *sign-to-atomic-formula* ‘ *A*) = *to-V* ‘ *A*› .
qed

lemma *atoms-to-V-AF*: ‹*atoms* (*AF-to-formula* (*Pair C A*)) = *to-V* ‘ (*fset A*)›
proof −
have ‹*atoms* (*AF-to-formula* (*Pair C A*)) = ⋃ (*atoms* ‘ *sign-to-atomic-formula* ‘ *fset A*)›
using *atoms-bigor set-maps-list-of-fset atoms-neg* **unfolding** *AF-to-formula-def*
by (*smt* (*z3*) *AF.sel*(*2*) *image-iff subsetI subset-antisym*)
also have ‹... = *to-V* ‘ (*fset A*)›
using *atoms-to-V* **by** *auto*
ultimately show ‹*atoms* (*AF-to-formula* (*Pair C A*)) = *to-V* ‘ (*fset A*)› **by** *simp*
qed

lemma *atoms-to-V-A-of*: ‹*atoms* (*AF-to-formula* 𝒞) = *to-V* ‘ (*fset* (*A-of* 𝒞))›
using *atoms-to-V-AF*
by (*metis AF.collapse*)

lemma *atoms-to-V-un*: ‹⋃(*atoms* ‘ *AF-to-formula* ‘ 𝒮) = ⋃{*to-V* ‘ *fset A* |*A*. *A* ∈ *A-of* ‘ 𝒮}›
proof −
have ‹(*x* ∈ (*atoms* ‘ *AF-to-formula* ‘ 𝒮)) = (*x* ∈ {*to-V* ‘ (*fset A*) |*A*. *A* ∈ *A-of* ‘ 𝒮})› **for** *x*
using *atoms-to-V-A-of* **by** *blast*
then show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *Collect-cong Sup-set-def UNION-singleton-eq-range mem-Collect-eq*)
qed

lemma *atoms-simp*: ‹$\bigcup$ (*atoms* ‘ (*AF-proj-to-formula-set S*)) = *to-V* ‘ $\bigcup$ (*fset* ‘ (*A-of* ‘ (*proj*$_\perp$ *S*)))›
proof −
have ‹$\bigcup$ (*atoms* ‘ (*AF-proj-to-formula-set S*)) = $\bigcup$ {*to-V* ‘ *fset A* | *A*. *A* ∈ *A-of* ‘ (*proj*$_\perp$ *S*)}›
unfolding *AF-proj-to-formula-set-def* **using** *atoms-to-V-un* **by** *auto*
also have ‹... = *to-V* ‘ $\bigcup$ (*fset* ‘ (*A-of* ‘ (*proj*$_\perp$ *S*)))›
by *blast*
finally show *?thesis* **.**
qed

lemma *val-from-interp*: ‹∀ $\mathcal{A}$. ∃ *J*. $\mathcal{A}$ = *to-valuation J*›
proof
fix $\mathcal{A}$
define *J-bare* **where** ‹*J-bare* = {*Pos a* |(*a*::′*v*). $\mathcal{A}$ *a*} ∪ {*Neg a* |*a*. ¬ $\mathcal{A}$ *a*}›
then have *interp-J-bare*: ‹*is-interpretation J-bare*›
unfolding *is-interpretation-def*
by *force*
then have *total-J-bare*: ‹*total* (*interp-of J-bare*)›
unfolding *total-def* **using** *J-bare-def*
by (*metis* (*mono-tags*, *lifting*) *Abs-propositional-interpretation-inverse Un-iff belong-to.rep-eq*
mem-Collect-eq to-V.simps)
define *J* **where** ‹*J* = *total-interp-of J-bare*›
have ‹$\mathcal{A}$ = *to-valuation J*›
proof
fix *a*::′*v*
show ‹$\mathcal{A}$ *a* = *to-valuation J a*›
using *J-def J-bare-def Abs-propositional-interpretation-inverse*
Abs-total-interpretation-inverse interp-J-bare total-J-bare to-valuation-def val-strip-pos
by *fastforce*
qed
then show ‹∃ *J*. $\mathcal{A}$ = *to-valuation J*›
by *fast*
qed

lemma *interp-from-val*: ‹∀ *J*. ∃ $\mathcal{A}$. $\mathcal{A}$ = *to-valuation J*›
unfolding *to-valuation-def* **by** *auto*

lemma *compactness-unsat*: ‹(¬ *sat* (*S*::′*v formula set*)) ⟷ (∃ *s*⊆*S*. *finite s* ∧ ¬ *sat s*)›
using *compactness*[*of S*] **unfolding** *fin-sat-def* **by** *blast*

lemma *never-enabled-finite-subset*:
‹∀ *J*. ¬ *enabled-set* $\mathcal{N}$ *J* ⟹ ∃ $\mathcal{N}'$⊆$\mathcal{N}$. *finite* $\mathcal{N}'$ ∧ (∀ *J*. ¬ *enabled-set* $\mathcal{N}'$ *J*)›
proof −
assume *not-enab-N*: ‹∀ *J*. ¬ *enabled-set* $\mathcal{N}$ *J*›

then have ‹¬ *sat* (*AF-assertions-to-formula-set* $\mathcal{N}$)›
unfolding *sat-def* **using** *equiv-enabled-assertions-sema*[*of* $\mathcal{N}$] *val-from-interp*
by *metis*
then obtain S' **where** S'*-sub*: ‹$S' \subseteq$ *AF-assertions-to-formula* ‘ $\mathcal{N}$› **and** S'*-fin*: ‹*finite* S'› **and**
S'*-unsat*: ‹¬ *sat* S'›
using *compactness-unsat* **unfolding** *AF-assertions-to-formula-set-def* **by** *metis*
obtain $\mathcal{N}'$ **where** N'*-sub*: ‹$\mathcal{N}' \subseteq \mathcal{N}$› **and** N'*-fin*: ‹*finite* $\mathcal{N}'$›
and S'*-im*: ‹$S' =$ *AF-assertions-to-formula* ‘ $\mathcal{N}'$›
using *finite-subset-image*[*OF* S'*-fin* S'*-sub*] **by** *blast*
have *not-enab-N'*: ‹$\forall J.$ ¬ *enabled-set* $\mathcal{N}'$ J›
using *equiv-enabled-assertions-sema*[*of* $\mathcal{N}'$] S'*-unsat* S'*-im interp-from-val*
unfolding *sat-def AF-assertions-to-formula-set-def* **by** *blast*
then show ‹$\exists \mathcal{N}' \subseteq \mathcal{N}$. *finite* $\mathcal{N}' \wedge (\forall J.$ ¬ *enabled-set* $\mathcal{N}'$ J)›
using N'*-sub* N'*-fin* **by** *auto*
qed

lemma *compactness-AF-proj*: ‹$(\forall J.$ ¬ $J \models_p 2$ $\mathcal{N}) \longleftrightarrow (\exists \mathcal{N}' \subseteq \mathcal{N}$. *finite* $\mathcal{N}' \wedge (\forall J.$ ¬ $J \models_p 2$ $\mathcal{N}'))$›
proof −
define $\mathcal{F}$ **where** ‹$\mathcal{F} =$ *AF-proj-to-formula-set* $\mathcal{N}$›
have ‹$(\forall J.$ ¬ $J \models_p 2$ $\mathcal{N}) \longleftrightarrow (\forall J. \exists F \in \mathcal{F}.$ ¬ *formula-semantics* (*to-valuation* J) F)›
by (*simp add*: $\mathcal{F}$*-def equiv-prop-entail2-sema2*)
also have
‹$(\forall J. \exists F \in \mathcal{F}.$ ¬ *formula-semantics* (*to-valuation* J) $F) \longleftrightarrow (\forall \mathcal{A}. \exists F \in \mathcal{F}.$ ¬ *formula-semantics* $\mathcal{A}$ F)›
using *val-from-interp* **by** *metis*
also have ‹$(\forall \mathcal{A}. \exists F \in \mathcal{F}.$ ¬ *formula-semantics* $\mathcal{A}$ $F) \longleftrightarrow$ (¬ *sat* $\mathcal{F}$)›
unfolding *sat-def* **by** *blast*
also have ‹(¬ *sat* $\mathcal{F}$) $\longleftrightarrow (\exists \mathcal{F}' \subseteq \mathcal{F}.$ *finite* $\mathcal{F}' \wedge$ ¬ *sat* $\mathcal{F}'$)›
using *compactness-unsat*[*of* $\mathcal{F}$] **.**
also have ‹$(\exists \mathcal{F}' \subseteq \mathcal{F}.$ *finite* $\mathcal{F}' \wedge$ ¬ *sat* $\mathcal{F}') \longleftrightarrow (\exists \mathcal{F}' \subseteq \mathcal{F}.$ *finite* $\mathcal{F}' \wedge (\forall \mathcal{A}. \exists F \in \mathcal{F}'.$ ¬ *formula-semantics* $\mathcal{A}$ F))›
unfolding *sat-def* **by** *auto*
also have ‹... $\longleftrightarrow (\exists \mathcal{F}' \subseteq \mathcal{F}.$ *finite* $\mathcal{F}' \wedge (\forall J. \exists F \in \mathcal{F}'.$ ¬ *formula-semantics* (*to-valuation* J) F))›
by (*metis val-from-interp*)
also have ‹... $\longleftrightarrow (\exists \mathcal{N}' \subseteq \mathcal{N}.$ *finite* $\mathcal{N}' \wedge (\forall J.$ ¬ $J \models_p 2$ $\mathcal{N}'))$›
proof
assume ‹$\exists \mathcal{F}' \subseteq \mathcal{F}.$ *finite* $\mathcal{F}' \wedge (\forall J. \exists F \in \mathcal{F}'.$ ¬ *formula-semantics* (*to-valuation* J) F)›
then obtain $\mathcal{F}'$ **where** F'*-sub*: ‹$\mathcal{F}' \subseteq \mathcal{F}$› **and** F'*-fin*: ‹*finite* $\mathcal{F}'$› **and**
F'*-unsat*: ‹$\forall J. \exists F \in \mathcal{F}'.$ ¬ *formula-semantics* (*to-valuation* J) F›
by *auto*
have ‹$\forall F \in \mathcal{F}'. \exists \mathcal{C} \in (proj_\perp\ \mathcal{N})$. *AF-to-formula* $\mathcal{C} = F$›
using F'*-sub* $\mathcal{F}$*-def* **unfolding** *AF-proj-to-formula-set-def* **by** *blast*
then obtain $\mathcal{N}'$ **where** F'*-is-map*: ‹$\mathcal{F}' =$ *AF-to-formula* ‘ $\mathcal{N}'$› **and** N'*-in-proj*:

‹$\mathcal{N}' \subseteq$ *proj*$_\perp$ $\mathcal{N}$› **and**
N'-fin: ‹*finite* $\mathcal{N}'$›
using *F'-fin*
by (*metis AF-proj-to-formula-set-def F'-sub* $\mathcal{F}$*-def finite-subset-image*)
have ‹*proj*$_\perp$ $\mathcal{N}' = \mathcal{N}'$›
using *N'-in-proj* **unfolding** *propositional-projection-def* **by** *blast*
then have *F'-is*: ‹$\mathcal{F}'$ = *AF-proj-to-formula-set* $\mathcal{N}'$›
unfolding *AF-proj-to-formula-set-def* **using** *F'-is-map* **by** *simp*
have *N'-sub*: ‹$\mathcal{N}' \subseteq \mathcal{N}$›
using *prop-proj-in N'-in-proj* **by** *auto*
have *N'-unsat*: ‹$\forall J.\ \neg\ J \models_p 2\ \mathcal{N}'$›
using *equiv-prop-entail2-sema2*[*of* - $\mathcal{N}'$] *F'-is F'-unsat*
by *blast*
show ‹$\exists \mathcal{N}' \subseteq \mathcal{N}.$ *finite* $\mathcal{N}' \wedge (\forall J.\ \neg\ J \models_p 2\ \mathcal{N}')$›
using *N'-sub N'-fin N'-unsat* **by** *blast*
next
assume ‹$\exists \mathcal{N}' \subseteq \mathcal{N}.$ *finite* $\mathcal{N}' \wedge (\forall J.\ \neg\ J \models_p 2\ \mathcal{N}')$›
then obtain $\mathcal{N}'$ **where** *N'-sub*: ‹$\mathcal{N}' \subseteq \mathcal{N}$› **and** *N'-fin*: ‹*finite* $\mathcal{N}'$› **and** *N'-unsat*: ‹$\forall J.\ \neg\ J \models_p 2\ \mathcal{N}'$›
by *auto*
define $\mathcal{F}'$ **where** ‹$\mathcal{F}'$ = *AF-proj-to-formula-set* $\mathcal{N}'$›
then have ‹$\mathcal{F}' \subseteq \mathcal{F}$›
using *N'-sub* **unfolding** $\mathcal{F}$*-def AF-proj-to-formula-set-def propositional-projection-def* **by** *blast*
moreover have ‹*finite* $\mathcal{F}'$›
using $\mathcal{F}'$*-def N'-fin* **unfolding** *AF-proj-to-formula-set-def propositional-projection-def* **by** *simp*
moreover have ‹$\forall J.\ \exists F \in \mathcal{F}'.\ \neg$ *formula-semantics* (*to-valuation J*) *F*›
using *N'-unsat equiv-prop-entail2-sema2*[*of* - $\mathcal{N}'$] **unfolding** $\mathcal{F}'$*-def* **by** *blast*
ultimately show ‹$\exists \mathcal{F}' \subseteq \mathcal{F}.$ *finite* $\mathcal{F}' \wedge (\forall J.\ \exists F \in \mathcal{F}'.\ \neg$ *formula-semantics* (*to-valuation J*) *F*)›
by *auto*
qed
finally show ‹$(\forall J.\ \neg\ J \models_p 2\ \mathcal{N}) \longleftrightarrow (\exists \mathcal{N}' \subseteq \mathcal{N}.$ *finite* $\mathcal{N}' \wedge (\forall J.\ \neg\ J \models_p 2\ \mathcal{N}'))$›
.
qed

lemma *prop-unsat-compactness*:
‹*propositionally-unsatisfiable A* $\Longrightarrow \exists\ B \subseteq A.$ *finite B* $\wedge$ *propositionally-unsatisfiable B*›
by (*meson compactness-AF-proj equiv-prop-entails propositionally-unsatisfiable-def*)

definition $\mathcal{E}$*-from* :: ‹('*f*, '*v*) *AF set* $\Rightarrow$ ('*f*, '*v*) *AF set*› **where**
‹$\mathcal{E}$*-from* $\mathcal{N} \equiv \{$*Pair bot* $\{|$*neg a*$|\}\ |a.\ \exists \mathcal{C} \in \mathcal{N}.\ a \in$ *fset* (*A-of* $\mathcal{C}$)$\}$›

lemma *prop-proj-*$\mathcal{E}$*-from*: ‹*proj*$_\perp$ ($\mathcal{E}$*-from* $\mathcal{N}$) = $\mathcal{E}$*-from* $\mathcal{N}$›
unfolding *propositional-projection-def* $\mathcal{E}$*-from-def* **by** *auto*

lemma *prop-proj-sub*: ‹*proj*$_\bot$ $\mathcal{N}$ = $\mathcal{N}$ ⟹ $\mathcal{N}'$ ⊆ $\mathcal{N}$ ⟹ *proj*$_\bot$ $\mathcal{N}'$ = $\mathcal{N}'$›
unfolding *propositional-projection-def* **by** *blast*

lemma *prop-proj-distrib*: ‹*proj*$_\bot$ (*A* ∪ *B*) = *proj*$_\bot$ *A* ∪ *proj*$_\bot$ *B*›
unfolding *propositional-projection-def* **by** *blast*

lemma *v-in-*$\mathcal{E}$: ‹*Pair bot* {|*Pos v*|} ∈ $\mathcal{E}$*-from* $\mathcal{N}$ ∨ *Pair bot* {|*Neg v*|} ∈ $\mathcal{E}$*-from* $\mathcal{N}$
⟹
∃$\mathcal{C}$∈$\mathcal{N}$. *v* ∈ *to-V* ‘ (*fset* (*A-of* $\mathcal{C}$))›
unfolding $\mathcal{E}$*-from-def* **by** (*smt* (*verit*, *ccfv-threshold*) *AF.sel*(2) *fthe-felem-eq image-iff*
mem-Collect-eq neg.simps(1) *neg.simps*(2) *to-V.elims to-V.simps*(1) *to-V.simps*(2))

lemma *a-in-*$\mathcal{E}$: ‹∃ *J*. *J* ⊨$_p$2 $\mathcal{E}$*-from* $\mathcal{N}$ ⟹ *Pair bot* {|*neg a*|} ∈ $\mathcal{E}$*-from* $\mathcal{N}$ ⟹
¬ (*Pair bot* {|*a*|} ∈ $\mathcal{E}$*-from* $\mathcal{N}$)›
proof
assume
e-sat: ‹∃ *J*. *J* ⊨$_p$2 $\mathcal{E}$*-from* $\mathcal{N}$› **and**
neg-a-in: ‹*Pair bot* {|*neg a*|} ∈ $\mathcal{E}$*-from* $\mathcal{N}$› **and**
a-in: ‹*AF.Pair bot* {|*a*|} ∈ $\mathcal{E}$*-from* $\mathcal{N}$›
obtain *J* **where** *J-sat-e*: ‹*J* ⊨$_p$2 $\mathcal{E}$*-from* $\mathcal{N}$›
using *e-sat* **by** *blast*
have *neg-a-in-J*: ‹*neg a* ∈ *total-strip J*›
using *a-in J-sat-e* **unfolding** *propositional-model2-def* $\mathcal{E}$*-from-def enabled-projection-def*
propositional-projection-def enabled-def **by** (*smt* (*verit*, *ccfv-SIG*) *AF.collapse AF.inject*
bot-fset.rep-eq empty-iff empty-subsetI finsert.rep-eq insert-subset mem-Collect-eq
neg.simps(1) *neg.simps*(2) *to-V.elims val-strip-neg val-strip-pos*)
have ‹*neg a* ∈ *total-strip J* ⟹ ¬ *J* ⊨$_p$2 $\mathcal{E}$*-from* $\mathcal{N}$›
using *neg-a-in J-sat-e enabled-def*
enabled-projection-def prop-proj-$\mathcal{E}$*-from propositional-model2-def* **by** *fastforce*
then show *False*
using *neg-a-in-J J-sat-e* **by** *blast*
qed

lemma *equiv-*$\mathcal{E}$*-enabled-*$\mathcal{N}$:
shows ‹*J* ⊨$_p$2 $\mathcal{E}$*-from* $\mathcal{N}$ ⟷ *enabled-set* $\mathcal{N}$ *J*›
unfolding *propositional-model2-def enabled-set-def enabled-def propositional-projection-def*
enabled-projection-def
proof
assume *empty-proj-E*: ‹{} = {*F-of* $\mathcal{C}$ |$\mathcal{C}$. $\mathcal{C}$ ∈ {$\mathcal{C}$ ∈ $\mathcal{E}$*-from* $\mathcal{N}$. *F-of* $\mathcal{C}$ = *bot*} ∧
fset (*A-of* $\mathcal{C}$) ⊆ *total-strip J*}›
have ‹∀$\mathcal{C}$∈$\mathcal{E}$*-from* $\mathcal{N}$. *F-of* $\mathcal{C}$ = *bot*› **using** $\mathcal{E}$*-from-def*[*of* $\mathcal{N}$] **by** *auto*
then have *a-in-E*: ‹∀$\mathcal{C}$∈$\mathcal{E}$*-from* $\mathcal{N}$. ∃ *a*∈*fset* (*A-of* $\mathcal{C}$). *a* ∉ *total-strip J*›
using *empty-proj-E* **by** *blast*
then have ‹∀$\mathcal{C}$∈$\mathcal{E}$*-from* $\mathcal{N}$. ∀ *a*∈*fset* (*A-of* $\mathcal{C}$). *a* ∉ *total-strip J*›
unfolding $\mathcal{E}$*-from-def* **by** *fastforce*
moreover have ‹∀$\mathcal{C}$∈$\mathcal{N}$. ∀ *a*∈*fset* (*A-of* $\mathcal{C}$). ∃$\mathcal{C}'$∈$\mathcal{E}$*-from* $\mathcal{N}$. *neg a* ∈ *fset* (*A-of* $\mathcal{C}'$)›

unfolding $\mathcal{E}$*-from-def* **by** *fastforce*
ultimately have ‹$\forall \mathcal{C}{\in}\mathcal{N}.\ \forall a{\in}$*fset* (*A-of* $\mathcal{C}$). $a \in$ *total-strip* J›
by *fastforce*
then show ‹$\forall \mathcal{C}{\in}\mathcal{N}.$ *fset* (*A-of* $\mathcal{C}$) $\subseteq$ *total-strip* J›
by *blast*
next
assume *enabled-*$\mathcal{C}$: ‹$\forall \mathcal{C}{\in}\mathcal{N}.$ *fset* (*A-of* $\mathcal{C}$) $\subseteq$ *total-strip* J›
have ‹$\forall \mathcal{C}{\in}\mathcal{E}$*-from* $\mathcal{N}.\ \forall a{\in}$*fset* (*A-of* $\mathcal{C}$). $\exists \mathcal{C}'{\in}\mathcal{N}.$ *neg* $a \in$ *fset* (*A-of* $\mathcal{C}'$)›
unfolding $\mathcal{E}$*-from-def*
by (*smt* (*verit*) *AF.exhaust-sel AF.inject bot-fset.rep-eq empty-iff finsert.rep-eq insertE*
is-Pos.cases mem-Collect-eq neg.simps)
then have ‹$\forall \mathcal{C}{\in}\mathcal{E}$*-from* $\mathcal{N}.\ \forall a{\in}$*fset* (*A-of* $\mathcal{C}$). $a \notin$ *total-strip* J›
using *enabled-*$\mathcal{C}$ **by** (*meson belong-to.rep-eq neg-prop-interp subsetD*)
then have ‹$\forall \mathcal{C}{\in}\mathcal{E}$*-from* $\mathcal{N}.$ ($\neg$ *fset* (*A-of* $\mathcal{C}$) $\subseteq$ *total-strip* J)›
using $\mathcal{E}$*-from-def* **by** *fastforce*
then show ‹$\{\} = \{$*F-of* $\mathcal{C}$ $|\mathcal{C}.\ \mathcal{C} \in \{\mathcal{C} \in \mathcal{E}$*-from* $\mathcal{N}.$ *F-of* $\mathcal{C} = bot\} \wedge$ *fset* (*A-of* $\mathcal{C}$) $\subseteq$ *total-strip* $J\}$›
by *blast*
qed

definition *AF-entails* :: ('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool* (**infix** $\models_{AF}$ *50*)
where
‹*AF-entails* $\mathcal{M}$ $\mathcal{N} \equiv (\forall J.$ (*enabled-set* $\mathcal{N}$ $J \longrightarrow \mathcal{M}$ $proj_J$ $J \models$ *F-of* ‘ $\mathcal{N}$))›

lemma *prop-unsat-to-AF-entails-bot*: ‹*propositionally-unsatisfiable* $\mathcal{M} \Longrightarrow \mathcal{M} \models_{AF}$ {*to-AF bot*}›
proof −
assume *prop-unsat-*$\mathcal{M}$: ‹*propositionally-unsatisfiable* $\mathcal{M}$›
then show ‹$\mathcal{M} \models_{AF}$ {*to-AF bot*}›
unfolding *AF-entails-def*
proof (*intro allI impI*)
fix J
assume ‹*enabled-set* {*to-AF bot*} J›
have ‹*bot* ∈ ($proj_\bot$ $\mathcal{M}$) $proj_J$ J›
using *prop-unsat-*$\mathcal{M}$
unfolding *propositionally-unsatisfiable-def propositional-model-def*
by *blast*
then have ‹*bot* ∈ $\mathcal{M}$ $proj_J$ J›
using *enabled-projection-def prop-proj-in*
by *fastforce*
then have ‹$\mathcal{M}$ $proj_J$ $J \models$ {*bot*}›
using *bot-entails-empty entails-subsets*
by (*meson empty-subsetI insert-subset*)
then show ‹$\mathcal{M}$ $proj_J$ $J \models$ *F-of* ‘ {*to-AF bot*}›
by (*auto simp add*: *F-of-to-AF*)
qed
qed

lemma ‹*enabled-set* {} *J*›
unfolding *enabled-set-def* **by** *blast*

lemma ‹(∀ *J*. ¬ (*enabled-set* $\mathcal{N}$ *J*)) ⟹ ($\mathcal{M}$ $\models_{AF}$ $\mathcal{N}$)›
unfolding *AF-entails-def* **by** *blast*

definition *AF-entails-sound* :: ('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool* (**infix** $\models s_{AF}$ *50*) **where**
‹*AF-entails-sound* $\mathcal{M}$ $\mathcal{N}$ ≡ (∀ *J*. (*enabled-set* $\mathcal{N}$ *J* ⟶
sound-cons.entails-neg ((*fml-ext* ‘ (*total-strip J*)) ∪ (*Pos* ‘ ($\mathcal{M}$ $proj_J$ *J*))) (*Pos* ‘ *F-of* ‘ $\mathcal{N}$)))›

lemma *distrib-proj*: ‹$\mathcal{M}$ ∪ $\mathcal{N}$ $proj_J$ *J* = ($\mathcal{M}$ $proj_J$ *J*) ∪ ($\mathcal{N}$ $proj_J$ *J*)›
unfolding *enabled-projection-def* **by** *auto*

lemma *distrib-proj-singleton*: ‹$\mathcal{M}$ ∪ {$\mathcal{C}$} $proj_J$ *J* = ($\mathcal{M}$ $proj_J$ *J*) ∪ ({$\mathcal{C}$} $proj_J$ *J*)›
unfolding *enabled-projection-def* **by** *auto*

lemma *enabled-union2*: ‹*enabled-set* ($\mathcal{M}$ ∪ $\mathcal{N}$) *J* ⟹ *enabled-set* $\mathcal{N}$ *J*›
unfolding *enabled-set-def* **by** *blast*

lemma *enabled-union1*: ‹*enabled-set* ($\mathcal{M}$ ∪ $\mathcal{N}$) *J* ⟹ *enabled-set* $\mathcal{M}$ *J*›
unfolding *enabled-set-def* **by** *blast*

lemma *finite-subset-image-strong*:
assumes *finite U* **and**
(∀ *C* ∈ *U*. (∃ *D* ∈ *W*. *P D* = *C* ∧ *Q D*))
shows ∃ *W'*⊆*W*. *finite W'* ∧ *U* = *P* ‘ *W'* ∧ (∀ *D'*∈ *W'*. *Q D'*)
using *assms*
proof (*induction U rule*: *finite-induct*)
case *empty*
then show *?case* **by** *blast*
next
case (*insert D' U*)
then obtain *C' W''* **where** *wpp-and-cp-assms*: *W''* ⊆ *W finite W'' U* = *P* ‘ *W''* ∀ *a* ∈ *W''*. *Q a*
C' ∈ *W P C'* = *D' Q C'*
by *auto*
define *W'* **where** *W'* = *insert C' W''*
then have ‹(*insert C' W'*) ⊆*W* ∧ *finite* (*insert C' W'*) ∧ *insert D' U* = *P* ‘ (*insert C' W'*) ∧
(∀ *a*∈(*insert C' W'*). *Q a*)›
using *wpp-and-cp-assms* **by** *blast*
then show *?case*
by *blast*
qed

lemma *all-to-ex*: ‹∀ *x*. *P x* ⟹ ∃ *x*. *P x*› **for** *P* **by** *blast*

lemma *three-skolems*:
assumes ‹$\bigwedge U.\ P\ U \implies \exists X\ Y\ Z.\ Q\ U\ X\ Y\ Z$›
shows ‹$(\bigwedge X\text{-}of\ Y\text{-}of\ Z\text{-}of.\ (\bigwedge U.\ P\ U \implies Q\ U\ (X\text{-}of\ U)\ (Y\text{-}of\ U)\ (Z\text{-}of\ U)) \implies thesis) \implies thesis$›
using *assms*
by *metis*

lemma *finite-subset-with-prop*:
assumes ‹$\exists Js.\ A = f\ `\ Js \wedge (\forall J{\in}Js.\ P\ J)$› **and**
‹*finite* C› **and**
‹$B = C \cap A$›
shows ‹$\exists Js.\ B = f\ `\ Js \wedge (\forall J{\in}Js.\ P\ J) \wedge$ *finite* Js›
proof −
have *B-fin*: ‹*finite* B› **using** *assms*(2) *assms*(3) **by** *simp*
have *B-sub*: ‹$B \subseteq A$› **using** *assms*(2) *assms*(3) **by** *auto*
obtain *Js* **where** *A-is*: ‹$A = f\ `\ Js$› **and** *P-Js*: ‹$\forall J{\in}Js.\ P\ J$›
using *assms*(1) **by** *blast*
then have ‹$\forall b{\in}B.\ \exists J{\in}Js.\ b = f\ J \wedge P\ J$›
using *B-sub* **by** *blast*
then obtain *Js′* **where** ‹$B = f\ `\ Js'$› **and** ‹*finite* Js'› ‹$\forall J{\in}Js'.\ P\ J$›
using *B-fin* **by** (*smt* (*verit*, *ccfv-threshold*) *B-sub assms*(1) *finite-subset-image subsetD*)
then show ‹$\exists Js.\ B = f\ `\ Js \wedge (\forall J{\in}Js.\ P\ J) \wedge$ *finite* Js›
by *blast*
qed

lemma *to-V-neg* [*simp*]: ‹*to-V* (*neg* a) = *to-V* a›
by (*metis is-Neg-to-V is-Pos-to-V neg.simps*(1) *neg.simps*(2) *to-V.simps*(1) *to-V.simps*(2))

sublocale *AF-cons-rel*: *consequence-relation to-AF bot AF-entails*
proof
show ‹$\{$*to-AF bot*$\} \models_{AF} \{\}$›
unfolding *to-AF-def AF-entails-def enabled-def enabled-projection-def*
using *bot-entails-empty* **by** *simp*
next
fix $\mathcal{C}$
show ‹$\{\mathcal{C}\} \models_{AF} \{\mathcal{C}\}$›
unfolding *to-AF-def AF-entails-def enabled-def enabled-projection-def enabled-set-def*
using *entails-reflexive* **by** *simp*
next
fix $\mathcal{M}\ \mathcal{N}\ \mathcal{P}\ \mathcal{Q}$
assume *m-in-n*: ‹$\mathcal{M} \subseteq \mathcal{N}$› **and**
p-in-q: ‹$\mathcal{P} \subseteq \mathcal{Q}$› **and**
m-entails-p: ‹$\mathcal{M} \models_{AF} \mathcal{P}$›
show ‹$\mathcal{N} \models_{AF} \mathcal{Q}$›
unfolding *to-AF-def AF-entails-def enabled-def enabled-projection-def enabled-set-def*
proof (*rule allI*, *rule impI*)
fix J

```
    assume q-enabled: ‹∀𝒞∈𝒬. fset (A-of 𝒞) ⊆ total-strip J›
    have ‹{F-of 𝒞 |𝒞. 𝒞 ∈ ℳ ∧ fset (A-of 𝒞) ⊆ total-strip J} ⊆
      {F-of 𝒞 |𝒞. 𝒞 ∈ 𝒩 ∧ fset (A-of 𝒞) ⊆ total-strip J}›
      using m-in-n by blast
    moreover have ‹F-of ‘ 𝒫 ⊆ F-of ‘ 𝒬›
      using p-in-q by blast
    ultimately show ‹{F-of 𝒞 |𝒞. 𝒞 ∈ 𝒩 ∧ fset (A-of 𝒞) ⊆ total-strip J} ⊨ F-of
‘ 𝒬›
      using m-entails-p  entails-subsets
        unfolding to-AF-def AF-entails-def enabled-def enabled-projection-def en-
abled-set-def
      by (metis (mono-tags, lifting) q-enabled p-in-q subset-iff)
  qed
next
  fix ℳ 𝒩 𝒞 ℳ' 𝒩'
  assume
    entails-c: ‹ℳ ⊨_AF 𝒩 ∪ {𝒞}› and
    c-entails: ‹ℳ' ∪ {𝒞} ⊨_AF 𝒩'›
  show ‹ℳ ∪ ℳ' ⊨_AF 𝒩 ∪ 𝒩'›
    unfolding AF-entails-def
  proof (intro allI impI)
    fix J
    assume enabled-n: ‹enabled-set (𝒩 ∪ 𝒩') J›
    {
      assume enabled-c: ‹enabled-set {𝒞} J›
      then have proj-enabled-c: ‹{𝒞} proj_J J = {F-of 𝒞}›
        unfolding enabled-projection-def using enabled-set-def by blast
      have cut-hyp1: ‹ℳ proj_J J ⊨ F-of ‘ 𝒩 ∪ {F-of 𝒞}›
       using entails-c enabled-n enabled-c unfolding AF-entails-def by (simp add:
enabled-set-def)
      have ‹(ℳ' ∪ {𝒞}) proj_J J ⊨ F-of ‘ 𝒩'›
          using c-entails enabled-union2[of 𝒩 𝒩' J, OF enabled-n] unfolding
AF-entails-def by simp
      then have cut-hyp2: ‹(ℳ' proj_J J) ∪ {F-of 𝒞} ⊨ F-of ‘ 𝒩'›
        using proj-enabled-c distrib-proj-singleton by metis
      have ‹ℳ ∪ ℳ' proj_J J ⊨ F-of ‘ (𝒩 ∪ 𝒩')›
      using entails-cut[OF cut-hyp1 cut-hyp2] distrib-proj by (simp add: image-Un)

    }
    moreover
    {
      assume not-enabled-c: ‹¬ enabled-set {𝒞} J›
      then have ‹ℳ' ∪ {𝒞} proj_J J = ℳ' proj_J J›
        unfolding enabled-projection-def enabled-set-def by auto
      then have ‹ℳ' proj_J J ⊨ F-of ‘ 𝒩'›
     using c-entails enabled-n unfolding AF-entails-def by (metis enabled-union2)

      then have ‹ℳ ∪ ℳ' proj_J J ⊨ F-of ‘ (𝒩 ∪ 𝒩')›
        using entails-subsets by (metis distrib-proj image-Un sup.cobounded2)
```

```
    }
    ultimately show ‹M ∪ M' proj_J J ⊨ F-of ' (N ∪ N')›
      by blast
    qed
next
  fix M N
  assume m-entails-n: ‹M ⊨_AF N›
  consider (NotEnabled) ‹∀ J. ¬ enabled-set N J› | (Enabled) ‹∃ J. enabled-set N
J› by blast
  then show ‹∃ M' N'. M' ⊆ M ∧ N' ⊆ N ∧ finite M' ∧ finite N' ∧ M' ⊨_AF
N'›
  proof cases
    case NotEnabled
    then obtain N'
where N'-sub: ‹N' ⊆ N› and N'-fin: ‹finite N'› and
      sub-not-enab: ‹∀ J. ¬ enabled-set N' J›
      using never-enabled-finite-subset[of N] by blast
    obtain M' where ‹M' ⊆ M› and ‹finite M'› and ‹M' ⊨_AF N'›
      using sub-not-enab unfolding AF-entails-def by blast
    then show ?thesis using N'-sub N'-fin by blast
  next
    case Enabled
    then obtain J' where J'-is: ‹enabled-set N J'› by auto
    {
      fix J
      assume enabled-N: ‹enabled-set N J›
      then have ‹M proj_J J ⊨ F-of ' N›
        using m-entails-n unfolding AF-entails-def by simp
      then obtain M' N' where mp-proj: ‹M' ⊆ M proj_J J› and
        np-proj: ‹N' ⊆ F-of ' N› and mp-fin: ‹finite M'› and np-fin: ‹finite N'›
and
        mp-entails-np: ‹M' ⊨ N'›
        using entails-compactness by metis

      have mp-with-f-of: ‹∀ C ∈ M'. ∃ C ∈ M. F-of C = C ∧ enabled C J›
        using mp-proj unfolding enabled-projection-def by blast
      have ‹∃ M'⊆ M. finite M' ∧ M' = F-of ' M' ∧ enabled-set M' J›
        using finite-subset-image-strong[OF mp-fin mp-with-f-of]
        unfolding enabled-set-def by blast
      then have m-fin-subset: ‹∃ M' ⊆ M. finite M' ∧ M' = M' proj_J J›
        unfolding enabled-projection-def enabled-set-def by blast

      have np-with-f-of: ‹∀ C ∈ N'. ∃ C ∈ N. F-of C = C›
        using np-proj unfolding enabled-projection-def by blast
      have n-fin-subset: ‹∃ N'⊆ N. finite N' ∧ N' = F-of ' N'›
        using finite-subset-image[OF np-fin np-proj] .

      obtain M' N' where m-n-subs: ‹M' ⊆ M› ‹N' ⊆ N› ‹finite M'› ‹finite
N'› ‹M' = M' proj_J J›
        ‹N' = F-of ' N'›
```

using *m-fin-subset n-fin-subset* **by** *blast*
then have *m-proj*: ‹$\mathcal{M}'$ *proj*$_J$ $J \models$ *F-of* ‘ $\mathcal{N}'$›
using *mp-entails-np* **by** *simp*

have *enabled-N'*: ‹*enabled-set* $\mathcal{N}'$ J›
using *enabled-N m-n-subs*(2) **unfolding** *enabled-set-def* **by** *blast*

let *?*$\mathcal{M}'$*-sel*$_J$ = ‹$\{\mathcal{C}.\ \mathcal{C} \in \mathcal{M}' \wedge$ *enabled* $\mathcal{C}$ $J\}$›
have ‹*?*$\mathcal{M}'$*-sel*$_J \subseteq \mathcal{M}'$› **by** *simp*
have ‹*finite* $(\bigcup\{$*fset* (*A-of* $\mathcal{C}$) $|\mathcal{C}.\ \mathcal{C} \in \mathcal{N}' \cup$ *?*$\mathcal{M}'$*-sel*$_J\})$›
using *m-n-subs*(3) *m-n-subs*(4) **by** *auto*
then obtain $A_{\mathcal{J}'}$ **where** *AJ-is*: ‹*fset* $A_{\mathcal{J}'} = \bigcup\{$*fset* (*A-of* $\mathcal{C}$) $|\mathcal{C}.\ \mathcal{C} \in \mathcal{N}' \cup$ *?*$\mathcal{M}'$*-sel*$_J\}$›
by (*smt* (*verit*, *best*) *fset-cases mem-Collect-eq*)
define $\mathcal{J}'$ **where** ‹$\mathcal{J}'$ = *Pair bot* $A_{\mathcal{J}'}$ ›
have ‹$\forall$ *a*∈*fset* (*A-of* $\mathcal{J}'$). $a \in_t J$›
proof
fix *a*
assume ‹$a \in$ *fset* (*A-of* $\mathcal{J}'$)›
then have ‹$\exists \mathcal{C} \in \mathcal{N}' \cup$ *?*$\mathcal{M}'$*-sel*$_J$. $a \in$ *fset* (*A-of* $\mathcal{C}$)›
unfolding $\mathcal{J}'$*-def* **using** *AJ-is* **by** *auto*
then show ‹$a \in_t J$›
using *enabled-N'* **unfolding** *enabled-set-def enabled-def belong-to-total-def belong-to-def*
by *auto*
qed
moreover have ‹*F-of* $\mathcal{J}'$ = *bot*›
unfolding $\mathcal{J}'$*-def*
by *simp*
moreover have ‹$\forall \mathcal{C} \in \mathcal{N}'$. *fset* (*A-of* $\mathcal{C}$) $\subseteq$ *fset* (*A-of* $\mathcal{J}'$)›
using *AJ-is* $\mathcal{J}'$*-def* **by** *auto*

ultimately have
‹$\exists \mathcal{M}'\ \mathcal{N}'\ \mathcal{J}'.\ \mathcal{M}' \subseteq \mathcal{M} \wedge \mathcal{N}' \subseteq \mathcal{N} \wedge$ *finite* $\mathcal{M}' \wedge$ *finite* $\mathcal{N}' \wedge \mathcal{M}'$ *proj*$_J$ $J \models$ *F-of* ‘ $\mathcal{N}' \wedge$
enabled-set $\mathcal{N}'$ $J \wedge$ *F-of* $\mathcal{J}'$ = *bot* $\wedge$ ($\forall$ *a*∈*fset* (*A-of* $\mathcal{J}'$). $a \in_t J$) $\wedge$
(*fset* (*A-of* $\mathcal{J}'$) = $\bigcup\{$*fset* (*A-of* $\mathcal{C}$) $|\mathcal{C}.\ \mathcal{C} \in \mathcal{N}' \cup \{\mathcal{C}.\ \mathcal{C} \in \mathcal{M}' \wedge$ *enabled* $\mathcal{C}$ $J\}\})$›
using *enabled-N' m-n-subs m-proj AJ-is* $\mathcal{J}'$*-def*
by (*metis* (*mono-tags*, *lifting*) *AF.sel*(2))
}

then obtain $\mathcal{M}'$*-of* $\mathcal{N}'$*-of* $\mathcal{J}'$*-of* **where**
fsets-from-J: ‹*enabled-set* $\mathcal{N}$ $J \Longrightarrow$ $\mathcal{M}'$*-of* $J \subseteq \mathcal{M} \wedge$ $\mathcal{N}'$*-of* $J \subseteq \mathcal{N} \wedge$ *finite* ($\mathcal{M}'$*-of* J) $\wedge$
finite ($\mathcal{N}'$*-of* J) $\wedge$ ($\mathcal{M}'$*-of* J) *proj*$_J$ $J \models$ *F-of* ‘ ($\mathcal{N}'$*-of* J) $\wedge$ *enabled-set* ($\mathcal{N}'$*-of* J) $J \wedge$
F-of ($\mathcal{J}'$*-of* J) = *bot* $\wedge$ ($\forall$ *a*∈*fset* (*A-of* ($\mathcal{J}'$*-of* J)). $a \in_t J$) $\wedge$

```
    (fset (A-of (𝒥′-of J)) = ⋃{fset (A-of 𝒞) |𝒞. 𝒞 ∈ (𝒩′-of J) ∪
      {𝒞. 𝒞 ∈ (ℳ′-of J) ∧ enabled 𝒞 J}})› for J
    using three-skolems[of λU. enabled-set 𝒩 U
      λJ ℳ′ 𝒩′ 𝒥′. ℳ′ ⊆ ℳ ∧ 𝒩′ ⊆ 𝒩 ∧ finite ℳ′ ∧ finite 𝒩′ ∧ ℳ′ proj_J J
⊨ F-of ‘ 𝒩′ ∧
      enabled-set 𝒩′ J ∧ F-of 𝒥′ = bot ∧ (∀ a∈fset (A-of 𝒥′). a ∈_t J) ∧
      (fset (A-of 𝒥′) = ⋃{fset (A-of 𝒞) |𝒞. 𝒞 ∈ 𝒩′ ∪ {𝒞. 𝒞 ∈ ℳ′ ∧ enabled 𝒞
J}})]
    by force

  let ?𝒥′-set = ‹{𝒥′-of J |J. enabled-set 𝒩 J}›
  have ex-Js: ‹∃ Js. ?𝒥′-set = 𝒥′-of ‘ Js ∧ (∀ J∈Js. enabled-set 𝒩 J)›
    by blast
  have proj-prop-J′: ‹proj_⊥ ?𝒥′-set = ?𝒥′-set›
    using fsets-from-J unfolding propositional-projection-def by blast
  let ?𝒩′-un = ‹⋃{𝒩′-of J |J. enabled-set 𝒩 J}›
  let ?ℳ′-un = ‹⋃{{𝒞. 𝒞 ∈ ℳ′-of J ∧ enabled 𝒞 J} |J. enabled-set 𝒩 J}›
  have A-of-enabled: ‹enabled-set 𝒩 J ⟹ (fset (A-of (𝒥′-of J)) =
    ⋃{fset (A-of 𝒞) |𝒞. 𝒞 ∈ (𝒩′-of J) ∪ {𝒞. 𝒞 ∈ (ℳ′-of J) ∧ enabled 𝒞 J}})›
for J
    using fsets-from-J by presburger
  have A-of-eq: ‹⋃ (fset ‘ A-of ‘ ?𝒥′-set) =
    ⋃ (fset ‘ A-of ‘ ?𝒩′-un) ∪ ⋃ (fset ‘ A-of ‘ ?ℳ′-un)›
  proof −
    have ‹⋃ (fset ‘ A-of ‘ ?𝒥′-set) = ⋃{fset (A-of (𝒥′-of J)) |J. enabled-set 𝒩
J}›
      by blast
    also have ‹... = ⋃{⋃{fset (A-of 𝒞) |𝒞. 𝒞 ∈
      (𝒩′-of J) ∪ {𝒞. 𝒞 ∈ (ℳ′-of J) ∧ enabled 𝒞 J}} |J. enabled-set 𝒩 J}›
      using A-of-enabled by (metis (no-types, lifting))
    also have ‹... = ⋃(fset ‘ A-of ‘ (?𝒩′-un ∪ ?ℳ′-un))› by blast
    finally show ‹⋃(fset ‘ A-of ‘ ?𝒥′-set) =
      ⋃(fset ‘ A-of ‘ ?𝒩′-un) ∪ ⋃ (fset ‘ A-of ‘ ?ℳ′-un)›
      by simp
  qed

  have ‹∀ J. ¬ (enabled-set 𝒩 J) ⟶ ¬ (J ⊨_p2 (ℰ-from 𝒩))›
    using equiv-ℰ-enabled-𝒩 by blast
  then have not-enab-case: ‹∀ J. ¬ (enabled-set 𝒩 J) ⟶ ¬ (J ⊨_p2 ?𝒥′-set ∪
(ℰ-from 𝒩))›
    using supset-not-model-p2 Un-upper2 by blast
  have ‹∀ J. enabled-set 𝒩 J ⟶ ¬ (J ⊨_p2 ?𝒥′-set)›
  proof (rule allI, rule impI, rule notI)
    fix J
    assume
      enab-N-loc: ‹enabled-set 𝒩 J› and
      entails-J: ‹(J ⊨_p2 ?𝒥′-set)›
    have A-ok: ‹fset (A-of (𝒥′-of J)) ⊆ total-strip J›
      using enab-N-loc fsets-from-J by force
```

then have ‹$proj_\bot$ {$\mathcal{J}'$-*of* J} $proj_J$ J = {*bot*}›
using *enab-N-loc fsets-from-J* **unfolding** *propositional-projection-def enabled-projection-def*
by (*simp add*: *enabled-def*)
then have ‹¬ $J \models_p 2$?$\mathcal{J}'$-*set*›
using *A-ok enab-N-loc* **unfolding** *propositional-model2-def enabled-def enabled-projection-def*
proj-prop-J' **by** *auto*
then show *False*
using *entails-J* **by** *auto*
qed
then have *enab-case*: ‹∀ J. (*enabled-set* $\mathcal{N}$ J) ⟶ ¬ ($J \models_p 2$?$\mathcal{J}'$-*set* ∪ ($\mathcal{E}$-*from* $\mathcal{N}$))›
using *supset-not-model-p2 Un-upper2* **by** *blast*
have ‹∀ J. ¬ ($J \models_p 2$ (?$\mathcal{J}'$-*set* ∪ ($\mathcal{E}$-*from* $\mathcal{N}$)))›
using *not-enab-case enab-case* **by** *blast*

then obtain $\mathcal{S}$ **where** *S-sub*: ‹$\mathcal{S}$ ⊆ ?$\mathcal{J}'$-*set* ∪ ($\mathcal{E}$-*from* $\mathcal{N}$)› **and** *S-fin*: ‹*finite* $\mathcal{S}$› **and**
S-unsat: ‹∀ J. ¬ $J \models_p 2$ $\mathcal{S}$›
using *compactness-AF-proj* **by** *meson*
have *E-sat*: ‹*sat* (*AF-proj-to-formula-set* ($\mathcal{E}$-*from* $\mathcal{N}$))›
unfolding *sat-def* **using** *J'-is equiv-$\mathcal{E}$-enabled-$\mathcal{N}$ equiv-prop-entail2-sema2* **by** *blast*
define $\mathcal{S}_\mathcal{J}$ **where** ‹$\mathcal{S}_\mathcal{J}$ = $\mathcal{S}$ ∩ ?$\mathcal{J}'$-*set*›
define $\mathcal{S}_\mathcal{E}$ **where** ‹$\mathcal{S}_\mathcal{E}$ = $\mathcal{S}$ ∩ ($\mathcal{E}$-*from* $\mathcal{N}$)›
define $\mathcal{S}_\mathcal{E}'$ **where** ‹$\mathcal{S}_\mathcal{E}'$ = {$\mathcal{C}$. $\mathcal{C}$ ∈ $\mathcal{S}_\mathcal{E}$ ∧ (*to-V* ‘ (*fset* (*A-of* $\mathcal{C}$))) ⊆ (*to-V* ‘ ⋃ (*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{J}$))}›
define $\mathcal{S}'$ **where** ‹$\mathcal{S}'$ = $\mathcal{S}_\mathcal{J}$ ∪ $\mathcal{S}_\mathcal{E}'$›
have *proj-S'*: ‹$proj_\bot$ $\mathcal{S}'$ = $\mathcal{S}'$›
using *proj-prop-J' prop-proj-$\mathcal{E}$-from S-sub prop-proj-sub prop-proj-distrib*
unfolding $\mathcal{S}'$-*def* $\mathcal{S}_\mathcal{J}$-*def* $\mathcal{S}_\mathcal{E}'$-*def* $\mathcal{S}_\mathcal{E}$-*def*
by (*smt* (*verit*) *Int-iff mem-Collect-eq subsetI*)
have *S-is*: ‹$\mathcal{S}$ = ($\mathcal{S}_\mathcal{E}$ − $\mathcal{S}_\mathcal{E}'$) ∪ $\mathcal{S}'$›
using *S-sub* $\mathcal{S}_\mathcal{J}$-*def* $\mathcal{S}_\mathcal{E}$-*def* $\mathcal{S}'$-*def* $\mathcal{S}_\mathcal{E}'$-*def* **by** *blast*
have *a-from-E-to-J*: ‹*neg* a ∈ ⋃ (*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{E}'$) ⟹ a ∈ ⋃ (*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{J}$)› **for** a
proof −
fix a
assume *nega-in*: ‹*neg* a ∈ ⋃ (*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{E}'$)›
then have ‹*to-V* (*neg* a) ∈ *to-V* ‘ ⋃ (*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{J}$)›
unfolding $\mathcal{S}_\mathcal{E}'$-*def* **by** *blast*
then have *a-or-nega-in*: ‹a ∈ ⋃ (*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{J}$) ∨ *neg* a ∈ ⋃ (*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{J}$)›
by (*smt* (*verit*) *imageE neg.simps*(*1*) *neg.simps*(*2*) *to-V.elims*)
obtain $\mathcal{C}$*1* **where** ‹$\mathcal{C}$*1* ∈ $\mathcal{E}$-*from* $\mathcal{N}$› **and** ‹*neg* a ∈ *fset* (*A-of* $\mathcal{C}$*1*)›
using *nega-in* **unfolding** $\mathcal{S}_\mathcal{E}'$-*def* $\mathcal{S}_\mathcal{E}$-*def* $\mathcal{E}$-*from-def* **by** *blast*
then obtain $\mathcal{C}$ **where** ‹$\mathcal{C}$ ∈ $\mathcal{N}$› **and** ‹a ∈ *fset* (*A-of* $\mathcal{C}$)›
unfolding $\mathcal{E}$-*from-def* **by** (*smt* (*verit*) *AF.sel*(*2*) *bot-fset.rep-eq empty-iff*

finsert.rep-eq
insert-iff mem-Collect-eq neg.simps(1) neg.simps(2) to-V.elims)
then have *in-N-in-J*: ‹$\forall J.\ (\mathit{enabled\text{-}set}\ \mathcal{N}\ J \longrightarrow a \in_t J)$›
using *in-total-to-strip* **unfolding** *enabled-set-def enabled-def* **by** *blast*
have ‹$b \in \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_{\mathcal{J}}) \Longrightarrow (\exists J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J \wedge b \in_t J)$› **for** *b*
proof −
have ‹$x \in \mathcal{S}_{\mathcal{J}} \Longrightarrow b\ |\in|\ \mathit{A\text{-}of}\ x \Longrightarrow \exists J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J \wedge b \in \mathit{total\text{-}strip}\ J$› **for** *x*
proof −
fix $\mathcal{C}2$
assume *C2-in*: ‹$\mathcal{C}2 \in \mathcal{S}_{\mathcal{J}}$› **and**
b-in: ‹$b \in \mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C}2)$›
obtain *J* **where** *enab-J*: ‹$\mathit{enabled\text{-}set}\ \mathcal{N}\ J$› **and** ‹$\mathcal{C}2 = \mathcal{J}'\text{-}\mathit{of}\ J$›
using *C2-in* **unfolding** $\mathcal{S}_{\mathcal{J}}$*-def* **by** *blast*
then have ‹$b \in \mathit{total\text{-}strip}\ J$›
using *b-in fsets-from-J* **by** *auto*
then show ‹$\exists J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J \wedge b \in \mathit{total\text{-}strip}\ J$›
using *enab-J* **by** *blast*
qed
then show ‹$b \in \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_{\mathcal{J}}) \Longrightarrow (\exists J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J \wedge b \in_t J)$›
by *clarsimp*
qed
then have ‹$\neg\ \ \mathit{neg}\ a \in \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_{\mathcal{J}})$›
using *in-N-in-J* **by** *fastforce*
then show ‹$a \in \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_{\mathcal{J}})$›
using *a-or-nega-in* **by** *blast*
qed
have *empty-inter-in-S*: ‹$\mathit{to\text{-}V}\ `\bigcup\ (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ (\mathcal{S}_{\mathcal{E}} - \mathcal{S}_{\mathcal{E}}')) \cap \mathit{to\text{-}V}\ `\bigcup\ (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}') = \{\}$›
proof (*rule ccontr*)
assume *contra*: ‹$\mathit{to\text{-}V}\ `\bigcup\ (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ (\mathcal{S}_{\mathcal{E}} - \mathcal{S}_{\mathcal{E}}')) \cap \mathit{to\text{-}V}\ `\bigcup\ (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}') \neq \{\}$›
then obtain *v* **where** *v-in1*: ‹$v \in \mathit{to\text{-}V}\ `\bigcup\ (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ (\mathcal{S}_{\mathcal{E}} - \mathcal{S}_{\mathcal{E}}'))$›
and *v-in2*: ‹$v \in \mathit{to\text{-}V}\ `\bigcup\ (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}')$› **by** *blast*
obtain $\mathcal{C}$ **where** *C-in*: ‹$\mathcal{C} \in \mathcal{S}_{\mathcal{E}} - \mathcal{S}_{\mathcal{E}}'$› **and** *v-in-C*: ‹$v \in \mathit{to\text{-}V}\ `\ (\mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C}))$›
using *v-in1* **by** *blast*
obtain *a* **where** *C-is1*: ‹$\mathcal{C} = \mathit{Pair}\ \mathit{bot}\ \{|a|\}$›
using *C-in* **unfolding** $\mathcal{S}_{\mathcal{E}}$*-def* $\mathcal{E}$*-from-def* **by** *blast*
then have *v-is*: ‹$v = \mathit{to\text{-}V}\ a$›
using *v-in-C* **by** *simp*
obtain $\mathcal{C}'$ **where** $\mathcal{C}'$*-in*: ‹$\mathcal{C}' \in \mathcal{S}'$› **and** *v-in-C'*: ‹$v \in \mathit{to\text{-}V}\ `\ (\mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C}'))$›
using *v-in2* **by** *blast*
then obtain a' **where** *v-is'*: ‹$v = \mathit{to\text{-}V}\ a'$› **and** a'*-in*: ‹$a' \in \mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C}')$›
by *blast*
consider (*J*) ‹$\mathcal{C}' \in \mathcal{S}_{\mathcal{J}}$› | (*E'*) ‹$\mathcal{C}' \in \mathcal{S}_{\mathcal{E}}'$›
using $\mathcal{C}'$*-in* $\mathcal{S}'$*-def* **by** *blast*
then show *False*
proof *cases*

```
      case J
      then have ‹to-V ‘ (fset (A-of C')) ⊆ (to-V ‘ ⋃ (fset ‘ A-of ‘ S_J))›
        by blast
      then have ‹to-V ‘ (fset (A-of C)) ⊆ (to-V ‘ ⋃ (fset ‘ A-of ‘ S_J))›
        using C-is1 v-in-C' v-is by auto
      then have ‹C ∈ S_E'›
        unfolding S_E'-def using C-in by blast
      then show False
        using C-in by blast
    next
      case E'
      then consider (a) ‹C' = Pair bot {|a|}› | (nega) ‹C' = Pair bot {|neg a|}›
        unfolding S_E'-def S_E-def E-from-def using v-in-C' v-is
          AF.sel(2) IntE empty-iff fset-simps(1) fset-simps(2) image-iff insert-iff
          mem-Collect-eq neg.simps(1) neg.simps(2) to-V.elims
        by (smt (verit, del-insts))
      then show False
      proof cases
        case a
        then show ?thesis
         using E' a-in-E Enabled equiv-E-enabled-N C-in C-is1 unfolding S_E-def
S_E'-def
          by blast
      next
        case nega
        have ‹C' ∈ E-from N›
          using E' unfolding S_E'-def S_E-def by auto
        moreover have ‹C ∈ E-from N›
          using C-in unfolding S_E-def by auto
        ultimately show ?thesis
          using a-in-E nega C-is1 Enabled equiv-E-enabled-N by blast
      qed
    qed
  qed
  then have empty-inter: ‹⋃ (atoms ‘ (AF-proj-to-formula-set (S_E − S_E'))) ∩
    ⋃ (atoms ‘ (AF-proj-to-formula-set S')) = {}›
    using atoms-simp proj-S' prop-proj-distrib prop-proj-sub
      by (smt (verit, ccfv-threshold) S-sub Un-subset-iff ‹S = S_E − S_E' ∪ S'›
proj-prop-J'
      prop-proj-E-from)
  have sat-rest: ‹sat (AF-proj-to-formula-set (S_E − S_E'))›
    using E-sat unfolding S_E'-def S_E-def AF-proj-to-formula-set-def
      propositional-projection-def sat-def by blast
  have S'-unsat: ‹∀ J. ¬ J ⊨_p 2 S'›
  using unsat-AF-simp[OF - sat-rest empty-inter] S-unsat equiv-prop-entail2-sema2
S-is
      val-from-interp unfolding sat-def by metis
  have ex-fin-Js: ‹∃ Js. S_J = J'-of ‘ Js ∧ (∀ J∈Js. enabled-set N J) ∧ finite Js›
    using finite-subset-with-prop[OF ex-Js S-fin S_J-def] .
```

then obtain *Js* **where** *Js-fin*: ‹*finite Js*› **and** *Js-enab*: ‹∀ *J*∈*Js*. *enabled-set* $\mathcal{N}$ *J*› **and**
Js-is: ‹$\mathcal{J}$'*-of* ‘ *Js* = $\mathcal{S}_{\mathcal{J}}$›
by *blast*

have *fin-inter*: ‹*finite* (⋃(*fset* ‘ *A-of* ‘ $\mathcal{S}_{\mathcal{J}}$) ∩ ⋃(*fset* ‘ *A-of* ‘ $\mathcal{N}$))›
proof
have ‹*finite* (⋃(*fset* ‘ *A-of* ‘ $\mathcal{S}_{\mathcal{J}}$))›
unfolding $\mathcal{S}_{\mathcal{J}}$*-def* **using** *S-fin image-Int-subset* **by** *simp*
then show ‹(*finite* (⋃(*fset* ‘ *A-of* ‘ $\mathcal{S}_{\mathcal{J}}$))) ∨ (*finite* (⋃ (*fset* ‘ *A-of* ‘ $\mathcal{N}$)))›
by *auto*
qed
have ‹∀ *a*∈(⋃(*fset* ‘ *A-of* ‘ $\mathcal{N}$)). ∃$\mathcal{C}$∈$\mathcal{N}$. *a* ∈ *fset* (*A-of* $\mathcal{C}$)›
by *blast*
then obtain *f* **where** *f-def*: ‹∀ *a*∈(⋃(*fset* ‘ *A-of* ‘ $\mathcal{N}$)). *f a* ∈ $\mathcal{N}$ ∧ *a* ∈ *fset* (*A-of* (*f a*))›
by *metis*

define $\mathcal{N}_{\mathcal{J}}$ **where** ‹$\mathcal{N}_{\mathcal{J}}$ = (*f* ‘ (⋃(*fset* ‘ *A-of* ‘ $\mathcal{S}_{\mathcal{J}}$) ∩ ⋃(*fset* ‘ *A-of* ‘ $\mathcal{N}$)))›
have *nj-fin*: ‹*finite* $\mathcal{N}_{\mathcal{J}}$›
unfolding $\mathcal{N}_{\mathcal{J}}$*-def* **using** *fin-inter* **by** *blast*
have *nj-sub*: ‹$\mathcal{N}_{\mathcal{J}}$ ⊆ $\mathcal{N}$›
unfolding $\mathcal{N}_{\mathcal{J}}$*-def* **using** *f-def* **by** *blast*
have *nj-as*: ‹(∀ *a*∈(⋃(*fset* ‘ *A-of* ‘ $\mathcal{S}_{\mathcal{J}}$))∩(⋃(*fset* ‘ *A-of* ‘ $\mathcal{N}$)).
a ∈ ⋃(*fset* ‘ *A-of* ‘ $\mathcal{N}_{\mathcal{J}}$))›
unfolding $\mathcal{N}_{\mathcal{J}}$*-def* **using** *f-def* **by** *fast*

define $\mathcal{M}'$ **where** ‹$\mathcal{M}'$ = ⋃{$\mathcal{M}'$*-of J* |*J*. *J* ∈ *Js*}›
define $\mathcal{N}'$ **where** ‹$\mathcal{N}'$ = ⋃{$\mathcal{N}'$*-of J* |*J*. *J* ∈ *Js*} ∪ $\mathcal{N}_{\mathcal{J}}$›
then have ‹$\mathcal{M}'$ ⊆ $\mathcal{M}$›
unfolding $\mathcal{M}'$*-def* **using** *fsets-from-J Js-enab* **by** *fast*
moreover have ‹$\mathcal{N}'$ ⊆ $\mathcal{N}$›
unfolding $\mathcal{N}'$*-def* **using** *fsets-from-J Js-enab nj-sub* **by** *fast*
moreover have ‹*finite* $\mathcal{M}'$›
unfolding $\mathcal{M}'$*-def* **using** *fsets-from-J Js-fin Js-enab* **by** *auto*
moreover have ‹*finite* $\mathcal{N}'$›
unfolding $\mathcal{N}'$*-def* **using** *fsets-from-J Js-fin Js-enab nj-fin* **by** *auto*
moreover have ‹$\mathcal{M}'$ ⊨$_{AF}$ $\mathcal{N}'$› **unfolding** *AF-entails-def*
proof (*rule allI*, *rule impI*)
fix *J*
assume *enab-N'*: ‹*enabled-set* $\mathcal{N}'$ *J*›
then have ‹*J* ⊨$_p$*2* $\mathcal{E}$*-from* $\mathcal{N}'$›
using *equiv-*$\mathcal{E}$*-enabled-*$\mathcal{N}$ **by** *auto*
moreover have ‹$\mathcal{S}_{\mathcal{E}}$' ⊆ $\mathcal{E}$*-from* $\mathcal{N}'$›
proof
fix $\mathcal{C}$
assume *C-in*: ‹$\mathcal{C}$ ∈ $\mathcal{S}_{\mathcal{E}}$'›
then obtain *a* **where** *C-is*: ‹$\mathcal{C}$ = *Pair bot* {|*neg a*|}›
unfolding $\mathcal{S}_{\mathcal{E}}$'*-def* $\mathcal{S}_{\mathcal{E}}$*-def* $\mathcal{E}$*-from-def* **by** *blast*

then have ‹*neg a* ∈ ⋃(*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{E}$′)›
using *C-in* **using** *image-iff* **by** *fastforce*
then have *a-in-SJ*: ‹*a* ∈ ⋃(*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{J}$)›
using *a-from-E-to-J* **by** *presburger*
have ‹∃$\mathcal{C}'$∈$\mathcal{N}$. *a* ∈ *fset* (*A-of* $\mathcal{C}'$)›
using *C-is C-in* **unfolding** $\mathcal{S}_\mathcal{E}$′*-def* $\mathcal{S}_\mathcal{E}$*-def* **by** (*smt* (*verit*, *ccfv-threshold*) *AF.sel*(2) *IntE J′-is* $\mathcal{E}$*-from-def a-in-*$\mathcal{E}$ *bot-fset.rep-eq empty-iff equiv-*$\mathcal{E}$*-enabled-*$\mathcal{N}$ *finsert.rep-eq insert-iff mem-Collect-eq to-V.elims to-V-neg*)
then have ‹*a* ∈ ⋃(*fset* ‘ *A-of* ‘ $\mathcal{N}$)›
by *blast*
then have ‹*a* ∈ ⋃(*fset* ‘ *A-of* ‘ $\mathcal{N}'$)›
using *nj-as a-in-SJ* **unfolding** $\mathcal{N}'$*-def* **by** *simp*
then show ‹$\mathcal{C}$ ∈ $\mathcal{E}$*-from* $\mathcal{N}'$›
using *C-is* **unfolding** $\mathcal{E}$*-from-def* **by** *blast*
qed
ultimately have ‹*J* ⊨$_p$*2* $\mathcal{S}_\mathcal{E}$′›
unfolding $\mathcal{S}_\mathcal{E}$′*-def* $\mathcal{S}_\mathcal{E}$*-def* **using** *subset-model-p2* **by** (*metis* (*no-types*, *lifting*))
then have ‹¬ *J* ⊨$_p$*2* $\mathcal{S}_\mathcal{J}$›
using *subset-not-model S′-unsat* **unfolding** $\mathcal{S}'$*-def* **by** *blast*
then have ‹∃ *J′*∈*Js*. *fset* (*A-of* (*J*′*-of J′*)) ⊆ *total-strip J*›
unfolding *propositional-model2-def* $\mathcal{S}_\mathcal{J}$*-def propositional-projection-def enabled-projection-def* **using** *Js-is*
by (*smt* (*verit*) *Collect-cong Set.empty-def* $\mathcal{S}_\mathcal{J}$*-def enabled-def image-iff mem-Collect-eq*)
then obtain *J′* **where** *J′-in*: ‹*J′* ∈ *Js*› **and** *A-of-J′-in*: ‹*fset* (*A-of* (*J*′*-of J′*)) ⊆ *total-strip J*›
by *blast*
then have *enab-nj′*: ‹*enabled-set* $\mathcal{N}$ *J′*›
using *Js-enab* **by** *blast*
then have ‹(*M*′*-of J′*) *proj*$_J$ *J′* ⊨ *F-of* ‘ (*N*′*-of J′*)›
using *fsets-from-J* **by** *auto*
moreover have ‹(*M*′*-of J′*) *proj*$_J$ *J′* ⊆ (*M*′*-of J′*) *proj*$_J$ *J*›
proof −
have ‹$\mathcal{C}$ ∈ *M*′*-of J′* ⟹ *enabled* $\mathcal{C}$ *J′* ⟹ *enabled* $\mathcal{C}$ *J*› **for** $\mathcal{C}$
proof −
assume *C-in*: ‹$\mathcal{C}$ ∈ *M*′*-of J′*› **and**
‹*enabled* $\mathcal{C}$ *J′*›
then have ‹*fset* (*A-of* $\mathcal{C}$) ⊆ *fset* (*A-of* (*J*′*-of J′*))›
using *fsets-from-J*[*OF enab-nj′*] **by** *blast*
then show ‹*enabled* $\mathcal{C}$ *J*›
using *A-of-J′-in* **unfolding** *enabled-def* **by** *auto*
qed
then have ‹($\mathcal{C}$ ∈ *M*′*-of J′* ∧ *enabled* $\mathcal{C}$ *J′*) ⟹ ($\mathcal{C}$ ∈ *M*′*-of J′* ∧ *enabled* $\mathcal{C}$ *J*)› **for** $\mathcal{C}$
by (*smt* (*verit*, *ccfv-threshold*))
then have ‹{$\mathcal{C}$. $\mathcal{C}$ ∈ *M*′*-of J′* ∧ *enabled* $\mathcal{C}$ *J′*} ⊆ {$\mathcal{C}$. $\mathcal{C}$ ∈ *M*′*-of J′* ∧ *enabled* $\mathcal{C}$ *J*}›

by *blast*
then show ‹($\mathcal{M}'$-*of* J') *proj*$_J$ $J' \subseteq$ ($\mathcal{M}'$-*of* J') *proj*$_J$ J›
unfolding *enabled-projection-def* **by** *blast*
qed
ultimately have *entails-one*: ‹($\mathcal{M}'$-*of* J') *proj*$_J$ $J \models$ *F-of* ‘ ($\mathcal{N}'$-*of* J')›
using *entails-subsets* **by** *blast*
have *subs-M*: ‹$\mathcal{M}'$-*of* J' *proj*$_J$ $J \subseteq \mathcal{M}'$ *proj*$_J$ J›
using J'-*in* **using** *enabled-projection-def* **unfolding** $\mathcal{M}'$-*def* **by** *auto*
have *subs-N*: ‹*F-of* ‘ ($\mathcal{N}'$-*of* J') $\subseteq$ *F-of* ‘ $\mathcal{N}'$›
using J'-*in* **unfolding** $\mathcal{N}'$-*def* **by** *blast*
show ‹$\mathcal{M}'$ *proj*$_J$ $J \models$ *F-of* ‘ $\mathcal{N}'$›
using *entails-subsets*[*OF subs-M subs-N entails-one*] **.**
qed

ultimately show ‹$\exists \mathcal{M}' \mathcal{N}'.\ \mathcal{M}' \subseteq \mathcal{M} \wedge \mathcal{N}' \subseteq \mathcal{N} \wedge$ *finite* $\mathcal{M}' \wedge$ *finite* $\mathcal{N}' \wedge \mathcal{M}' \models_{AF} \mathcal{N}'$›
by *blast*
qed
qed

sublocale *neg-ext-sound-cons-rel*: *consequence-relation Pos bot sound-cons.entails-neg*
using *sound-cons.ext-cons-rel* **by** *simp*

lemma *AF-ext-sound-cons-rel*: ‹*consequence-relation* (*to-AF bot*) *AF-entails-sound*›
proof (*standard*)
show ‹{*to-AF bot*} $\models s_{AF}$ {}›
unfolding *AF-entails-sound-def enabled-def enabled-projection-def*
proof (*rule allI*, *rule impI*)
fix J
assume ‹*enabled-set* {} J›
have *bot-in*: ‹{*Pos bot*} $\subseteq$ *Pos* ‘ {C. C = *F-of* (*to-AF bot*) $\wedge$ *fset* (*A-of* (*to-AF bot*)) $\subseteq$ *total-strip* J}›
unfolding *to-AF-def* **by** *simp*
have ‹*sound-cons.entails-neg* (*fml-ext* ‘ *total-strip* $J \cup$
Pos ‘ {C. C = *F-of* (*to-AF bot*) $\wedge$ *fset* (*A-of* (*to-AF bot*)) $\subseteq$ *total-strip* J}) {}›
using *sound-cons.bot-entails-empty sound-cons.entails-subsets bot-in*
by (*smt* (*verit*, *ccfv-threshold*) *AF.sel*(*2*) *Un-iff bot-fset.rep-eq*
consequence-relation.bot-entails-empty consequence-relation.entails-subsets empty-subsetI
image-iff mem-Collect-eq sound-cons.ext-cons-rel subset-eq to-AF-def)
then show ‹*fml-ext* ‘ *total-strip* $J \cup$
Pos ‘ {*F-of* $\mathcal{C}$ |$\mathcal{C}$. $\mathcal{C} \in$ {*to-AF bot*} $\wedge$ *fset* (*A-of* $\mathcal{C}$) $\subseteq$ *total-strip* J} $\models s_\sim$ *Pos* ‘ *F-of* ‘ {}›
by *clarsimp*
qed
next
fix $\mathcal{C}$ :: ('*f*, '*v*) *AF*

have ‹*Pos* ‘ {*F-of Ca* |*Ca*. *Ca* ∈ {$\mathcal{C}$} ∧ *fset* (*A-of Ca*) ⊆ *total-strip J*} ⊆ (*Pos* ‘ *F-of* ‘ {$\mathcal{C}$})›
by *auto*
show ‹{$\mathcal{C}$} $\models s_{AF}$ {$\mathcal{C}$}›
unfolding *AF-entails-sound-def enabled-def enabled-projection-def enabled-set-def*
proof (*rule allI*, *rule impI*)
fix *J*
assume *a-of-C-in*: ‹∀$\mathcal{C}$∈{$\mathcal{C}$}. *fset* (*A-of* $\mathcal{C}$) ⊆ *total-strip J*›
have ‹*sound-cons.entails-neg* (*Pos* (*F-of* $\mathcal{C}$) ▷ *fml-ext* ‘ *total-strip J*) {*Pos* (*F-of* $\mathcal{C}$)}›
using *sound-cons.entails-reflexive*[*of F-of* $\mathcal{C}$]
by (*smt* (*verit*, *best*) *Set.insert-mono bot.extremum consequence-relation.entails-reflexive consequence-relation.entails-subsets sound-cons.ext-cons-rel*)
then show ‹*fml-ext* ‘ *total-strip J* ∪
Pos ‘ {*F-of* $\mathcal{C}'$ |$\mathcal{C}'$. $\mathcal{C}'$ ∈ {$\mathcal{C}$} ∧ *fset* (*A-of* $\mathcal{C}'$) ⊆ *total-strip J*} $\models s_{\sim}$ *Pos* ‘ *F-of* ‘ {$\mathcal{C}$}›
using *a-of-C-in* **by** *clarsimp*
qed
next
fix $\mathcal{M}$ $\mathcal{N}$ $\mathcal{P}$ $\mathcal{Q}$
assume *m-in-n*: ‹$\mathcal{M}$ ⊆ $\mathcal{N}$› **and**
p-in-q: ‹$\mathcal{P}$ ⊆ $\mathcal{Q}$› **and**
m-entails-p: ‹$\mathcal{M}$ $\models s_{AF}$ $\mathcal{P}$›
show ‹$\mathcal{N}$ $\models s_{AF}$ $\mathcal{Q}$›
unfolding *AF-entails-sound-def enabled-def enabled-projection-def enabled-set-def*
proof (*rule allI*, *rule impI*)
fix *J*
assume *q-enabled*: ‹∀$\mathcal{C}$∈$\mathcal{Q}$. *fset* (*A-of* $\mathcal{C}$) ⊆ *total-strip J*›
have ‹{*F-of* $\mathcal{C}$ |$\mathcal{C}$. $\mathcal{C}$ ∈ $\mathcal{M}$ ∧ *fset* (*A-of* $\mathcal{C}$) ⊆ *total-strip J*} ⊆
{*F-of* $\mathcal{C}$ |$\mathcal{C}$. $\mathcal{C}$ ∈ $\mathcal{N}$ ∧ *fset* (*A-of* $\mathcal{C}$) ⊆ *total-strip J*}›
using *m-in-n* **by** *blast*
moreover have ‹*F-of* ‘ $\mathcal{P}$ ⊆ *F-of* ‘ $\mathcal{Q}$›
using *p-in-q* **by** *blast*
ultimately show ‹*sound-cons.entails-neg* (*fml-ext* ‘ *total-strip J* ∪
Pos ‘ {*F-of* $\mathcal{C}$ |$\mathcal{C}$. $\mathcal{C}$ ∈ $\mathcal{N}$ ∧ *fset* (*A-of* $\mathcal{C}$) ⊆ *total-strip J*}) (*Pos* ‘ *F-of* ‘ $\mathcal{Q}$)›
using *m-entails-p sound-cons.entails-subsets m-in-n p-in-q q-enabled*
unfolding *AF-entails-sound-def enabled-def enabled-projection-def enabled-set-def*
by (*smt* (*z3*) *Un-iff consequence-relation.entails-subsets image-iff mem-Collect-eq sound-cons.ext-cons-rel subset-eq*)
qed
next
fix $\mathcal{M}$ $\mathcal{N}$ $\mathcal{C}$ $\mathcal{M}'$ $\mathcal{N}'$
assume
entails-c: ‹$\mathcal{M}$ $\models s_{AF}$ $\mathcal{N}$ ∪ {$\mathcal{C}$}› **and**
c-entails: ‹$\mathcal{M}'$ ∪ {$\mathcal{C}$} $\models s_{AF}$ $\mathcal{N}'$›
show ‹$\mathcal{M}$ ∪ $\mathcal{M}'$ $\models s_{AF}$ $\mathcal{N}$ ∪ $\mathcal{N}'$›
unfolding *AF-entails-sound-def*
proof (*intro allI impI*)
fix *J*

assume *enabled-n*: ‹*enabled-set* $(\mathcal{N} \cup \mathcal{N}')$ J›
{
assume *enabled-c*: ‹*enabled-set* $\{\mathcal{C}\}$ J›
then have *proj-enabled-c*: ‹$\{\mathcal{C}\}$ $proj_J$ $J = \{F\text{-}of\ \mathcal{C}\}$›
unfolding *enabled-projection-def* **using** *enabled-set-def* **by** *blast*
have ‹*sound-cons.entails-neg* (*fml-ext* ‘ *total-strip* $J \cup Pos$ ‘ $(\mathcal{M}\ proj_J\ J)$)
(*Pos* ‘ *F-of* ‘ $(\mathcal{N} \cup \{\mathcal{C}\})$)›
using *entails-c enabled-n enabled-c* **unfolding** *AF-entails-sound-def*
by (*metis Un-iff enabled-set-def*)
then have *cut-hyp1*: ‹*sound-cons.entails-neg* (*fml-ext* ‘ *total-strip* $J \cup Pos$ ‘ $(\mathcal{M}\ proj_J\ J)$)
(*Pos* ‘ *F-of* ‘ $\mathcal{N} \cup \{Pos\ (F\text{-}of\ \mathcal{C})\}$)›
by *force*
have ‹*sound-cons.entails-neg* (*fml-ext* ‘ *total-strip* $J \cup Pos$ ‘ $(\mathcal{M}' \cup \{\mathcal{C}\}\ proj_J\ J)$)
(*Pos* ‘ *F-of* ‘ $\mathcal{N}'$)›
using *c-entails enabled-n enabled-union2* **unfolding** *AF-entails-sound-def* **by** *blast*
then have *cut-hyp2*: ‹*sound-cons.entails-neg*
(*fml-ext* ‘ *total-strip* $J \cup Pos$ ‘ $(\mathcal{M}'\ proj_J\ J) \cup \{Pos\ (F\text{-}of\ \mathcal{C})\})$ (*Pos* ‘ *F-of* ‘ $\mathcal{N}'$)›
by (*metis Un-empty-right Un-insert-right distrib-proj image-insert proj-enabled-c*)
have ‹*sound-cons.entails-neg* (*fml-ext* ‘ *total-strip* $J \cup Pos$ ‘ $(\mathcal{M} \cup \mathcal{M}'\ proj_J\ J)$)
(*Pos* ‘ *F-of* ‘ $(\mathcal{N} \cup \mathcal{N}')$)›
using *neg-ext-sound-cons-rel.entails-cut*[*OF cut-hyp1 cut-hyp2*] *distrib-proj*[*of* $\mathcal{M}$ $\mathcal{M}'$ J] *image-Un* **by** (*smt* (*verit*, *del-insts*) *Un-commute Un-left-absorb Un-left-commute*)
}
moreover
{
assume *not-enabled-c*: ‹¬ *enabled-set* $\{\mathcal{C}\}$ J›
then have ‹$\mathcal{M}' \cup \{\mathcal{C}\}\ proj_J\ J = \mathcal{M}'\ proj_J\ J$›
unfolding *enabled-projection-def enabled-set-def* **by** *auto*
then have ‹*sound-cons.entails-neg* (*fml-ext* ‘ *total-strip* $J \cup Pos$ ‘ $(\mathcal{M}'\ proj_J\ J)$)
(*Pos* ‘ *F-of* ‘ $\mathcal{N}'$)›
using *c-entails enabled-n enabled-union2* **unfolding** *AF-entails-sound-def* **by** *metis*
then have ‹*sound-cons.entails-neg* (*fml-ext* ‘ *total-strip* $J \cup Pos$ ‘ $(\mathcal{M} \cup \mathcal{M}'\ proj_J\ J)$)
(*Pos* ‘ *F-of* ‘ $(\mathcal{N} \cup \mathcal{N}')$)›
using *neg-ext-sound-cons-rel.entails-subsets*
by (*smt* (*verit*, *del-insts*) *Un-iff distrib-proj image-Un subsetI*)
}
ultimately
show ‹*sound-cons.entails-neg* (*fml-ext* ‘ *total-strip* $J \cup Pos$ ‘ $(\mathcal{M} \cup \mathcal{M}'\ proj_J\ J)$)
(*Pos* ‘ *F-of* ‘ $(\mathcal{N} \cup \mathcal{N}')$)›

```
    by blast
  qed
next
  fix 𝓜 𝓝
  assume m-entails-n: ‹𝓜 ⊨s_AF 𝓝›
  consider (NotEnabled) ‹∀ J. ¬ enabled-set 𝓝 J› | (Enabled) ‹∃ J. enabled-set 𝓝
J› by blast
  then show ‹∃ M' N'. M' ⊆ 𝓜 ∧ N' ⊆ 𝓝 ∧ finite M' ∧ finite N' ∧ M' ⊨s_AF
N'›
  proof cases
    case NotEnabled
    then obtain 𝓝' where N'-sub: ‹𝓝' ⊆ 𝓝› and N'-fin: ‹finite 𝓝'› and
      sub-not-enab: ‹∀ J. ¬ enabled-set 𝓝' J›
      using never-enabled-finite-subset[of 𝓝] by blast
    obtain 𝓜' where ‹𝓜' ⊆ 𝓜› and ‹finite 𝓜'› and ‹𝓜' ⊨s_AF 𝓝'›
      using sub-not-enab unfolding AF-entails-sound-def by blast
    then show ?thesis using N'-sub N'-fin by blast
  next
    case Enabled
    then obtain J' where J'-is: ‹enabled-set 𝓝 J'› by auto
    {
      fix J
      assume enabled-N: ‹enabled-set 𝓝 J›
      then have ‹sound-cons.entails-neg (fml-ext ‘ total-strip J ∪ Pos ‘ (𝓜 proj_J
J)) (Pos ‘ F-of ‘ 𝓝)›
        using m-entails-n unfolding AF-entails-sound-def by blast
       then obtain MJ' N' where mj-in: ‹MJ' ⊆ fml-ext ‘ total-strip J ∪ Pos ‘
(𝓜 proj_J J)› and
        np-proj: ‹N' ⊆ Pos ‘ F-of ‘ 𝓝› and mjp-fin: ‹finite MJ'› and np-fin: ‹finite
N'› and
        mjp-entails-np: ‹sound-cons.entails-neg  MJ' N'›
        using neg-ext-sound-cons-rel.entails-compactness by metis

      define M' where M' = MJ' ∩ Pos ‘ (𝓜 proj_J J)
      then have mp-fin: ‹finite M'›
        using mjp-fin by auto
      have mp-with-f-of: ‹∀ C ∈ M'. ∃ 𝒞 ∈ 𝓜. Pos (F-of 𝒞) = C ∧ enabled 𝒞 J›
        using mj-in unfolding enabled-projection-def M'-def by blast
      have ‹∃ 𝓜'⊆ 𝓜. finite 𝓜' ∧ M' = Pos ‘ F-of ‘ 𝓜' ∧ enabled-set 𝓜' J›
        using finite-subset-image-strong[of M' 𝓜 (λx. Pos (F-of x)) λx. enabled x
J, OF mp-fin mp-with-f-of]
        unfolding enabled-set-def by blast
      then have ex-mp: ‹∃ 𝓜'⊆𝓜. finite 𝓜' ∧ Pos ‘ (𝓜' proj_J J) = M'›
        unfolding enabled-projection-def enabled-set-def by blast
      then obtain 𝓜' where mp-props: ‹𝓜' ⊆ 𝓜› ‹finite 𝓜'› ‹Pos ‘ (𝓜' proj_J
J) = M'› by auto

      let ?𝓜'-sel_J = ‹{𝒞. 𝒞 ∈ 𝓜' ∧ enabled 𝒞 J}›
      have ‹?𝓜'-sel_J ⊆ 𝓜'› by simp
```

have ‹*finite* ($\bigcup$ {*fset* (*A-of* $\mathcal{C}$) |$\mathcal{C}$. $\mathcal{C}$ ∈ *?*$\mathcal{M}'$*-sel*$_J$})›
using *mp-props* **by** *auto*
then obtain $\mathcal{A}_{\mathcal{M}'}$ **where** *AM-is*: ‹*fset* $\mathcal{A}_{\mathcal{M}'}$ = ($\bigcup$ {*fset* (*A-of* $\mathcal{C}$) |$\mathcal{C}$. $\mathcal{C}$ ∈ *?*$\mathcal{M}'$*-sel*$_J$})›
using *fin-set-fset* **by** *fastforce*
then have *AM-in-J*: ‹*fset* $\mathcal{A}_{\mathcal{M}'}$ ⊆ *total-strip J*›
unfolding *enabled-def* **by** *blast*
define J' **where** J' = (MJ' ∩ *fml-ext* ‘ *total-strip J*)
then have *jp-fin*: ‹*finite* J'›
using *mjp-fin* **by** *blast*
then obtain $\mathcal{J}'$ **where** *jp-props*: $\mathcal{J}'$ ⊆ *total-strip J fml-ext* ‘ $\mathcal{J}'$ = J' *finite* $\mathcal{J}'$
using J'*-def* **by** (*metis Int-lower2 finite-subset-image*)
then obtain $\mathcal{A}_{\mathcal{J}'}$ **where** *AJ-is*: ‹*fset* $\mathcal{A}_{\mathcal{J}'}$ = $\mathcal{J}'$›
using *fin-set-fset* **by** *blast*
then have *AJ-in-J*: ‹*fset* $\mathcal{A}_{\mathcal{J}'}$ ⊆ *total-strip J*›
using *jp-props* **by** *auto*
define $\mathcal{J}_f{}'$ **where** ‹$\mathcal{J}_f{}'$ = *Pair bot* ($\mathcal{A}_{\mathcal{J}'}$ |∪| $\mathcal{A}_{\mathcal{M}'}$)›
then have *Jf-in*: ‹*fset* (*A-of* $\mathcal{J}_f{}'$) ⊆ *total-strip J*› **using** *AM-in-J AJ-in-J* **by** *simp*
have *Jf-bot*: ‹*F-of* $\mathcal{J}_f{}'$ = *bot*› **using** $\mathcal{J}_f{}'$*-def* **by** *auto*
have *AM-in-Jf*: ‹∀$\mathcal{C}$∈$\mathcal{M}'$. *enabled* $\mathcal{C}$ *J* ⟶ *fset* (*A-of* $\mathcal{C}$) ⊆ *fset* (*A-of* $\mathcal{J}_f{}'$)›
using $\mathcal{J}_f{}'$*-def AM-is* **by** *auto*

have *np-with-f-of*: ‹∀ C ∈ N'. ∃$\mathcal{C}$ ∈ $\mathcal{N}$. *Pos* (*F-of* $\mathcal{C}$) = C›
using *np-proj* **unfolding** *enabled-projection-def* **by** *blast*
have *n-fin-subset*: ‹∃$\mathcal{N}'$⊆ $\mathcal{N}$. *finite* $\mathcal{N}'$ ∧ N' = *Pos* ‘ *F-of* ‘ $\mathcal{N}'$›
using *finite-subset-image*[*OF np-fin, of* λ*x*. *Pos* (*F-of x*) $\mathcal{N}$] *np-proj* **by** *auto*
then obtain $\mathcal{N}'$ **where** *np-props*: ‹$\mathcal{N}'$ ⊆ $\mathcal{N}$› ‹*finite* $\mathcal{N}'$› ‹N' = *Pos* ‘ *F-of* ‘ $\mathcal{N}'$›
by *blast*
have *enab-np*: ‹*enabled-set* $\mathcal{N}'$ *J*›
using *enabled-N np-props* **unfolding** *enabled-set-def* **by** *blast*

have *mjp-is*: ‹MJ' = *Pos* ‘ ($\mathcal{M}'$ *proj*$_J$ *J*) ∪ *fml-ext* ‘ $\mathcal{J}'$›
using *mj-in* M'*-def* J'*-def mp-props jp-props* **by** *auto*
have ‹*sound-cons.entails-neg* ((*Pos* ‘ ($\mathcal{M}'$ *proj*$_J$ *J*))∪ *fml-ext* ‘ $\mathcal{J}'$) (*Pos* ‘ *F-of* ‘ $\mathcal{N}'$)›
using *np-props mjp-entails-np* **unfolding** *mjp-is* **by** *blast*
then have *fin-entail*: ‹*sound-cons.entails-neg* ((*Pos* ‘ ($\mathcal{M}'$ *proj*$_J$ *J*))∪ *fml-ext* ‘ (*fset* (*A-of* $\mathcal{J}_f{}'$))) (*Pos* ‘ *F-of* ‘ $\mathcal{N}'$)›
using *neg-ext-sound-cons-rel.entails-subsets*[*of*
(*Pos* ‘ ($\mathcal{M}'$ *proj*$_J$ *J*))∪ *fml-ext* ‘ $\mathcal{J}'$ (*Pos* ‘ ($\mathcal{M}'$ *proj*$_J$ *J*))∪ *fml-ext* ‘ (*fset* (*A-of* $\mathcal{J}_f{}'$))
Pos ‘ *F-of* ‘ $\mathcal{N}'$ *Pos* ‘ *F-of* ‘ $\mathcal{N}'$] *AJ-is* **unfolding** $\mathcal{J}_f{}'$*-def* **by** (*simp add*: *image-Un subsetI*)

have ‹∃$\mathcal{M}'$ $\mathcal{N}'$ $\mathcal{J}'$. $\mathcal{M}'$ ⊆ $\mathcal{M}$ ∧ *fset* (*A-of* $\mathcal{J}'$) ⊆ *total-strip J* ∧ $\mathcal{N}'$ ⊆ $\mathcal{N}$ ∧
finite $\mathcal{M}'$ ∧ *finite* $\mathcal{N}'$ ∧ *F-of* $\mathcal{J}'$ = *bot* ∧ *enabled-set* $\mathcal{N}'$ *J* ∧

$(\forall \mathcal{C}{\in}\mathcal{M}'.\ \mathit{enabled}\ \mathcal{C}\ J \longrightarrow \mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C}) \subseteq \mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{J}')) \wedge$
$\mathit{sound\text{-}cons.entails\text{-}neg}\ ((\mathit{Pos}\ `\ (\mathcal{M}'\ \mathit{proj}_J\ J)) \cup \mathit{fml\text{-}ext}\ `\ (\mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{J}')))$
$(\mathit{Pos}\ `\ \mathit{F\text{-}of}\ `\ \mathcal{N}')\rangle$
using *mp-props np-props fin-entail enab-np Jf-in Jf-bot* $\mathcal{J}_f$*'-def AJ-is AM-is AM-in-Jf AF.sel(2)* **by** *blast*
}

then obtain $\mathcal{M}'$*-of* $\mathcal{N}'$*-of* $\mathcal{J}'$*-of* **where**
fsets-from-J: $\langle \mathit{enabled\text{-}set}\ \mathcal{N}\ J \Longrightarrow \mathcal{M}'\text{-}\mathit{of}\ J \subseteq \mathcal{M} \wedge \mathit{fset}\ (\mathit{A\text{-}of}\ (\mathcal{J}'\text{-}\mathit{of}\ J)) \subseteq \mathit{total\text{-}strip}\ J \wedge \mathcal{N}'\text{-}\mathit{of}\ J \subseteq \mathcal{N} \wedge$
$\mathit{finite}\ (\mathcal{M}'\text{-}\mathit{of}\ J) \wedge \mathit{finite}\ (\mathcal{N}'\text{-}\mathit{of}\ J) \wedge \mathit{F\text{-}of}\ (\mathcal{J}'\text{-}\mathit{of}\ J) = \mathit{bot} \wedge \mathit{enabled\text{-}set}\ (\mathcal{N}'\text{-}\mathit{of}\ J)\ J \wedge$
$(\forall \mathcal{C}{\in}(\mathcal{M}'\text{-}\mathit{of}\ J).\ \mathit{enabled}\ \mathcal{C}\ J \longrightarrow \mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C}) \subseteq \mathit{fset}\ (\mathit{A\text{-}of}\ (\mathcal{J}'\text{-}\mathit{of}\ J))) \wedge$
$\mathit{sound\text{-}cons.entails\text{-}neg}\ (\mathit{Pos}\ `\ (\mathcal{M}'\text{-}\mathit{of}\ J\ \mathit{proj}_J\ J) \cup \mathit{fml\text{-}ext}\ `\ (\mathit{fset}\ (\mathit{A\text{-}of}\ (\mathcal{J}'\text{-}\mathit{of}\ J))))\ (\mathit{Pos}\ `\ \mathit{F\text{-}of}\ `\ \mathcal{N}'\text{-}\mathit{of}\ J)\ \rangle$ **for** J
by *metis*

let $?\mathcal{J}'\text{-}\mathit{set} = \langle\{\mathcal{J}'\text{-}\mathit{of}\ J\ |J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J\}\rangle$
have *ex-Js*: $\langle\exists\, \mathit{Js}.\ ?\mathcal{J}'\text{-}\mathit{set} = \mathcal{J}'\text{-}\mathit{of}\ `\ \mathit{Js} \wedge (\forall J{\in}\mathit{Js}.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J)\rangle$
by *blast*
have *proj-prop-J'*: $\langle \mathit{proj}_\perp\ ?\mathcal{J}'\text{-}\mathit{set} = ?\mathcal{J}'\text{-}\mathit{set}\rangle$
using *fsets-from-J* **unfolding** *propositional-projection-def* **by** *blast*
let $?\mathcal{N}'\text{-}\mathit{un} = \langle\bigcup\{\mathcal{N}'\text{-}\mathit{of}\ J\ |J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J\}\rangle$
let $?\mathcal{M}'\text{-}\mathit{un} = \langle\bigcup\{\{\mathcal{C}.\ \mathcal{C} \in \mathcal{M}'\text{-}\mathit{of}\ J \wedge \mathit{enabled}\ \mathcal{C}\ J\}\ |J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J\}\rangle$

have $\langle\forall J.\ \neg\ (\mathit{enabled\text{-}set}\ \mathcal{N}\ J) \longrightarrow \neg\ (J \models_p 2\ (\mathcal{E}\text{-}\mathit{from}\ \mathcal{N}))\rangle$
using *equiv-*$\mathcal{E}$*-enabled-*$\mathcal{N}$ **by** *blast*
then have *not-enab-case*: $\langle\forall J.\ \neg\ (\mathit{enabled\text{-}set}\ \mathcal{N}\ J) \longrightarrow \neg\ (J \models_p 2\ ?\mathcal{J}'\text{-}\mathit{set} \cup (\mathcal{E}\text{-}\mathit{from}\ \mathcal{N}))\rangle$
using *supset-not-model-p2 Un-upper2* **by** *blast*
have $\langle\forall J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J \longrightarrow \neg\ (J \models_p 2\ ?\mathcal{J}'\text{-}\mathit{set})\rangle$
proof (*rule allI*, *rule impI*, *rule notI*)
fix J
assume
enab-N-loc: $\langle \mathit{enabled\text{-}set}\ \mathcal{N}\ J\rangle$ **and**
entails-J: $\langle(J \models_p 2\ ?\mathcal{J}'\text{-}\mathit{set})\rangle$
have *A-ok*: $\langle \mathit{fset}\ (\mathit{A\text{-}of}\ (\mathcal{J}'\text{-}\mathit{of}\ J)) \subseteq \mathit{total\text{-}strip}\ J\rangle$
using *enab-N-loc fsets-from-J* **by** *force*
then have $\langle \mathit{proj}_\perp\ \{\mathcal{J}'\text{-}\mathit{of}\ J\}\ \mathit{proj}_J\ J = \{\mathit{bot}\}\rangle$
using *enab-N-loc fsets-from-J* **unfolding** *propositional-projection-def enabled-projection-def*
by (*simp add*: *enabled-def*)
then have $\langle\neg\ J \models_p 2\ ?\mathcal{J}'\text{-}\mathit{set}\rangle$
using *A-ok enab-N-loc* **unfolding** *propositional-model2-def enabled-def enabled-projection-def*
proj-prop-J' **by** *auto*
then show *False*
using *entails-J* **by** *auto*
qed

then have *enab-case*: ‹$\forall J.\ (\mathit{enabled\text{-}set}\ \mathcal{N}\ J) \longrightarrow \neg\ (J \models_p 2\ ?\mathcal{J}'\text{-}\mathit{set} \cup (\mathcal{E}\text{-}\mathit{from}\ \mathcal{N}))$›
using *supset-not-model-p2 Un-upper2* **by** *blast*
have ‹$\forall J.\ \neg\ (J \models_p 2\ (?\mathcal{J}'\text{-}\mathit{set} \cup (\mathcal{E}\text{-}\mathit{from}\ \mathcal{N})))$›
using *not-enab-case enab-case* **by** *blast*

then obtain $\mathcal{S}$ **where** *S-sub*: ‹$\mathcal{S} \subseteq ?\mathcal{J}'\text{-}\mathit{set} \cup (\mathcal{E}\text{-}\mathit{from}\ \mathcal{N})$› **and** *S-fin*: ‹*finite* $\mathcal{S}$› **and**
S-unsat: ‹$\forall J.\ \neg\ J \models_p 2\ \mathcal{S}$›
using *compactness-AF-proj* **by** *meson*
have *E-sat*: ‹*sat* (*AF-proj-to-formula-set* (*$\mathcal{E}$-from* $\mathcal{N}$))›
unfolding *sat-def* **using** *J'-is equiv-$\mathcal{E}$-enabled-$\mathcal{N}$ equiv-prop-entail2-sema2* **by** *blast*
define $\mathcal{S}_\mathcal{J}$ **where** ‹$\mathcal{S}_\mathcal{J} = \mathcal{S} \cap ?\mathcal{J}'\text{-}\mathit{set}$›
define $\mathcal{S}_\mathcal{E}$ **where** ‹$\mathcal{S}_\mathcal{E} = \mathcal{S} \cap (\mathcal{E}\text{-}\mathit{from}\ \mathcal{N})$›
define $\mathcal{S}_\mathcal{E}'$ **where** ‹$\mathcal{S}_\mathcal{E}' = \{\mathcal{C}.\ \mathcal{C} \in \mathcal{S}_\mathcal{E} \wedge (\mathit{to\text{-}V}\ `\ (\mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C}))) \subseteq (\mathit{to\text{-}V}\ `\ \bigcup (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{J}))\}$›
define $\mathcal{S}'$ **where** ‹$\mathcal{S}' = \mathcal{S}_\mathcal{J} \cup \mathcal{S}_\mathcal{E}'$›
have *proj-S'*: ‹$\mathit{proj}_\perp\ \ \mathcal{S}' = \mathcal{S}'$›
using *proj-prop-J' prop-proj-$\mathcal{E}$-from S-sub prop-proj-sub prop-proj-distrib*
unfolding *$\mathcal{S}'$-def $\mathcal{S}_\mathcal{J}$-def $\mathcal{S}_\mathcal{E}'$-def $\mathcal{S}_\mathcal{E}$-def*
by (*smt* (*verit*) *Int-iff mem-Collect-eq subsetI*)
have *S-is*: ‹$\mathcal{S} = (\mathcal{S}_\mathcal{E} - \mathcal{S}_\mathcal{E}') \cup \mathcal{S}'$›
using *S-sub $\mathcal{S}_\mathcal{J}$-def $\mathcal{S}_\mathcal{E}$-def $\mathcal{S}'$-def $\mathcal{S}_\mathcal{E}'$-def* **by** *blast*
have *a-from-E-to-J*: ‹$\mathit{neg}\ a \in \bigcup (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{E}') \Longrightarrow a \in \bigcup (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{J})$› **for** *a*
proof −
fix *a*
assume *nega-in*: ‹$\mathit{neg}\ a \in \bigcup (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{E}')$›
then have ‹$\mathit{to\text{-}V}\ (\mathit{neg}\ a) \in \mathit{to\text{-}V}\ `\ \bigcup (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{J})$›
unfolding *$\mathcal{S}_\mathcal{E}'$-def* **by** *blast*
then have *a-or-nega-in*: ‹$a \in \bigcup (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{J}) \vee \mathit{neg}\ a \in \bigcup (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{J})$›
by (*smt* (*verit*) *imageE neg.simps*(*1*) *neg.simps*(*2*) *to-V.elims*)
obtain $\mathcal{C}1$ **where** ‹$\mathcal{C}1 \in \mathcal{E}\text{-}\mathit{from}\ \mathcal{N}$› **and** ‹$\mathit{neg}\ a \in \mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C}1)$›
using *nega-in* **unfolding** *$\mathcal{S}_\mathcal{E}'$-def $\mathcal{S}_\mathcal{E}$-def $\mathcal{E}$-from-def* **by** *blast*
then obtain $\mathcal{C}$ **where** ‹$\mathcal{C} \in \mathcal{N}$› **and** ‹$a \in \mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C})$›
unfolding *$\mathcal{E}$-from-def* **by** (*smt* (*verit*) *AF.sel*(*2*) *bot-fset.rep-eq empty-iff finsert.rep-eq*
insert-iff mem-Collect-eq neg.simps(*1*) *neg.simps*(*2*) *to-V.elims*)
then have *in-N-in-J*: ‹$\forall J.\ (\mathit{enabled\text{-}set}\ \mathcal{N}\ J \longrightarrow a \in_t J)$›
using *in-total-to-strip* **unfolding** *enabled-set-def enabled-def* **by** *blast*
have ‹$b \in \bigcup (\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{J}) \Longrightarrow (\exists J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J \wedge b \in_t J)$› **for** *b*
proof −
have ‹$x \in \mathcal{S}_\mathcal{J} \Longrightarrow b\ |\in|\ \mathit{A\text{-}of}\ x \Longrightarrow \exists J.\ \mathit{enabled\text{-}set}\ \mathcal{N}\ J \wedge b \in \mathit{total\text{-}strip}\ J$› **for** *x*
proof −
fix $\mathcal{C}2$
assume *C2-in*: ‹$\mathcal{C}2 \in \mathcal{S}_\mathcal{J}$› **and**

```
      b-in: ‹b ∈ fset (A-of 𝒞2)›
     obtain J where enab-J: ‹enabled-set 𝒩 J› and ‹𝒞2 = 𝒥'-of J›
      using C2-in unfolding 𝒮_𝒥-def by blast
     then have ‹b ∈ total-strip J›
      using b-in fsets-from-J by (meson basic-trans-rules(31))
     then show ‹∃ J. enabled-set 𝒩 J ∧ b ∈ total-strip J›
      using enab-J by blast
   qed
   then show ‹b ∈ ⋃ (fset ‘ A-of ‘ 𝒮_𝒥) ⟹ ∃ J. enabled-set 𝒩 J ∧ b ∈_t J›
    by clarsimp
  qed
  then have ‹¬ neg a ∈ ⋃(fset ‘ A-of ‘ 𝒮_𝒥)›
   using in-N-in-J by fastforce
  then show ‹a ∈ ⋃(fset ‘ A-of ‘ 𝒮_𝒥)›
   using a-or-nega-in by blast
 qed
 have empty-inter-in-S: ‹to-V ‘ ⋃ (fset ‘ A-of ‘ (𝒮_ℰ − 𝒮_ℰ')) ∩ to-V ‘ ⋃ (fset ‘
A-of ‘ 𝒮') = {}›
 proof (rule ccontr)
  assume contra: ‹to-V ‘ ⋃ (fset ‘ A-of ‘ (𝒮_ℰ − 𝒮_ℰ')) ∩ to-V ‘ ⋃ (fset ‘ A-of
‘ 𝒮') ≠ {}›
  then obtain v where v-in1: ‹v ∈ to-V ‘ ⋃ (fset ‘ A-of ‘ (𝒮_ℰ − 𝒮_ℰ'))›
   and v-in2: ‹v ∈ to-V ‘ ⋃ (fset ‘ A-of ‘ 𝒮')› by blast
  obtain 𝒞 where C-in: ‹𝒞 ∈ 𝒮_ℰ − 𝒮_ℰ'› and v-in-C: ‹v ∈ to-V ‘ (fset (A-of
𝒞))›
   using v-in1 by blast
  obtain a where C-is1: ‹𝒞 = Pair bot {|a|}›
   using C-in unfolding 𝒮_ℰ-def ℰ-from-def by blast
  then have v-is: ‹v = to-V a›
   using v-in-C by simp
  obtain 𝒞' where C'-in: ‹𝒞' ∈ 𝒮'› and v-in-C': ‹v ∈ to-V ‘ (fset (A-of 𝒞'))›
   using v-in2 by blast
  then obtain a' where v-is': ‹v = to-V a'› and a'-in: ‹a' ∈ fset (A-of 𝒞')›
   by blast
  consider (J) ‹𝒞' ∈ 𝒮_𝒥› | (E') ‹𝒞' ∈ 𝒮_ℰ'›
   using C'-in 𝒮'-def by blast
  then show False
  proof cases
   case J
   then have ‹to-V ‘ (fset (A-of 𝒞')) ⊆ (to-V ‘ ⋃ (fset ‘ A-of ‘ 𝒮_𝒥))›
    by blast
   then have ‹to-V ‘ (fset (A-of 𝒞)) ⊆ (to-V ‘ ⋃ (fset ‘ A-of ‘ 𝒮_𝒥))›
    using C-is1 v-in-C' v-is by auto
   then have ‹𝒞 ∈ 𝒮_ℰ'›
    unfolding 𝒮_ℰ'-def using C-in by blast
   then show False
    using C-in by blast
  next
   case E'
```

then consider (a) ‹$\mathcal{C}' = Pair\ bot\ \{|a|\}$› | $(nega)$ ‹$\mathcal{C}' = Pair\ bot\ \{|neg\ a|\}$›
unfolding $\mathcal{S}_\mathcal{E}'$-*def* $\mathcal{S}_\mathcal{E}$-*def* $\mathcal{E}$-*from-def* **using** *v-in-C' v-is*
AF.sel(2) *IntE empty-iff fset-simps*(1) *fset-simps*(2) *image-iff insert-iff*
mem-Collect-eq neg.simps(1) *neg.simps*(2) *to-V.elims*
by (*smt* (*verit*, *del-insts*))
then show *False*
proof *cases*
case *a*
then show *?thesis*
using E' *a-in-*$\mathcal{E}$ *Enabled equiv-*$\mathcal{E}$*-enabled-*$\mathcal{N}$ *C-in C-is1* **unfolding** $\mathcal{S}_\mathcal{E}$-*def* $\mathcal{S}_\mathcal{E}'$-*def*
by *blast*
next
case *nega*
have ‹$\mathcal{C}' \in \mathcal{E}$-*from* $\mathcal{N}$›
using E' **unfolding** $\mathcal{S}_\mathcal{E}'$-*def* $\mathcal{S}_\mathcal{E}$-*def* **by** *auto*
moreover have ‹$\mathcal{C} \in \mathcal{E}$-*from* $\mathcal{N}$›
using *C-in* **unfolding** $\mathcal{S}_\mathcal{E}$-*def* **by** *auto*
ultimately show *?thesis*
using *a-in-*$\mathcal{E}$ *nega C-is1 Enabled equiv-*$\mathcal{E}$*-enabled-*$\mathcal{N}$ **by** *blast*
qed
qed
qed
then have *empty-inter*: ‹$\bigcup$ (*atoms* ‘ (*AF-proj-to-formula-set* ($\mathcal{S}_\mathcal{E}$ − $\mathcal{S}_\mathcal{E}'$))) ∩
$\bigcup$ (*atoms* ‘ (*AF-proj-to-formula-set* $\mathcal{S}'$)) = {}›
using *atoms-simp proj-S' prop-proj-distrib prop-proj-sub*
by (*smt* (*verit*, *ccfv-threshold*) *S-sub Un-subset-iff* ‹$\mathcal{S} = \mathcal{S}_\mathcal{E}$ − $\mathcal{S}_\mathcal{E}' \cup \mathcal{S}'$› *proj-prop-J'*
prop-proj-$\mathcal{E}$*-from*)
have *sat-rest*: ‹*sat* (*AF-proj-to-formula-set* ($\mathcal{S}_\mathcal{E}$ − $\mathcal{S}_\mathcal{E}'$))›
using *E-sat* **unfolding** $\mathcal{S}_\mathcal{E}'$-*def* $\mathcal{S}_\mathcal{E}$-*def AF-proj-to-formula-set-def*
propositional-projection-def sat-def **by** *blast*
have *S'-unsat*: ‹$\forall J.\ \neg\ J \models_p 2\ \mathcal{S}'$›
using *unsat-AF-simp*[*OF - sat-rest empty-inter*] *S-unsat equiv-prop-entail2-sema2 S-is*
val-from-interp **unfolding** *sat-def* **by** *metis*
have *ex-fin-Js*: ‹$\exists$ *Js*. $\mathcal{S}_\mathcal{J} = \mathcal{J}'$-*of* ‘ *Js* ∧ ($\forall$ *J*∈*Js*. *enabled-set* $\mathcal{N}$ *J*) ∧ *finite Js*›
using *finite-subset-with-prop*[*OF ex-Js S-fin* $\mathcal{S}_\mathcal{J}$-*def*] **.**
then obtain *Js* **where** *Js-fin*: ‹*finite Js*› **and** *Js-enab*: ‹$\forall$ *J*∈*Js*. *enabled-set* $\mathcal{N}$ *J*› **and**
Js-is: ‹$\mathcal{J}'$-*of* ‘ *Js* = $\mathcal{S}_\mathcal{J}$›
by *blast*

have *fin-inter*: ‹*finite* ($\bigcup$(*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{J}$) ∩ $\bigcup$(*fset* ‘ *A-of* ‘ $\mathcal{N}$))›
proof
have ‹*finite* ($\bigcup$(*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{J}$))›
unfolding $\mathcal{S}_\mathcal{J}$-*def* **using** *S-fin image-Int-subset* **by** *simp*
then show ‹(*finite* ($\bigcup$(*fset* ‘ *A-of* ‘ $\mathcal{S}_\mathcal{J}$))) ∨ (*finite* ($\bigcup$ (*fset* ‘ *A-of* ‘ $\mathcal{N}$)))›
by *auto*

qed
have ‹$\forall\, a{\in}(\bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{N}))$. $\exists \mathcal{C}{\in}\mathcal{N}.\ a \in \mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C})$›
by *blast*
then obtain f **where** *f-def*: ‹$\forall\, a{\in}(\bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{N}))$. $f\ a \in \mathcal{N} \wedge a \in \mathit{fset}\ (\mathit{A\text{-}of}\ (f\ a))$›
by *metis*
define $\mathcal{N}_\mathcal{J}$ **where** ‹$\mathcal{N}_\mathcal{J} = (f\ `\ (\bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{J}) \cap \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{N})))$›
have *nj-fin*: ‹*finite* $\mathcal{N}_\mathcal{J}$›
unfolding $\mathcal{N}_\mathcal{J}$*-def* **using** *fin-inter* **by** *blast*
have *nj-sub*: ‹$\mathcal{N}_\mathcal{J} \subseteq \mathcal{N}$›
unfolding $\mathcal{N}_\mathcal{J}$*-def* **using** *f-def* **by** *blast*
have *nj-as*: ‹$(\forall\, a{\in}(\bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{J}))\cap(\bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{N}))$.
$a \in \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{N}_\mathcal{J}))$›
unfolding $\mathcal{N}_\mathcal{J}$*-def* **using** *f-def* **by** *fast*

define $\mathcal{M}'$ **where** ‹$\mathcal{M}' = \bigcup\{\mathcal{M}'\text{-}\mathit{of}\ J\ |J.\ J \in \mathit{Js}\}$›
define $\mathcal{N}'$ **where** ‹$\mathcal{N}' = \bigcup\{\mathcal{N}'\text{-}\mathit{of}\ J\ |J.\ J \in \mathit{Js}\} \cup \mathcal{N}_\mathcal{J}$›
then have ‹$\mathcal{M}' \subseteq \mathcal{M}$›
unfolding $\mathcal{M}'$*-def* **using** *fsets-from-J Js-enab* **by** *fast*
moreover have ‹$\mathcal{N}' \subseteq \mathcal{N}$›
unfolding $\mathcal{N}'$*-def* **using** *fsets-from-J Js-enab nj-sub* **by** *fast*
moreover have ‹*finite* $\mathcal{M}'$›
unfolding $\mathcal{M}'$*-def* **using** *fsets-from-J Js-fin Js-enab* **by** *auto*
moreover have ‹*finite* $\mathcal{N}'$›
unfolding $\mathcal{N}'$*-def* **using** *fsets-from-J Js-fin Js-enab nj-fin* **by** *auto*
moreover have ‹$\mathcal{M}' \models s_{AF}\ \mathcal{N}'$› **unfolding** *AF-entails-sound-def*
proof (*rule allI*, *rule impI*)
fix J
assume *enab-N'*: ‹*enabled-set* $\mathcal{N}'$ J›
then have ‹$J \models_p 2\ \mathcal{E}$*-from* $\mathcal{N}'$›
using *equiv-*$\mathcal{E}$*-enabled-*$\mathcal{N}$ **by** *auto*
moreover have ‹$\mathcal{S}_\mathcal{E}{}' \subseteq \mathcal{E}$*-from* $\mathcal{N}'$›
proof
fix $\mathcal{C}$
assume *C-in*: ‹$\mathcal{C} \in \mathcal{S}_\mathcal{E}{}'$›
then obtain a **where** *C-is*: ‹$\mathcal{C} = \mathit{Pair}\ \mathit{bot}\ \{|\mathit{neg}\ a|\}$›
unfolding $\mathcal{S}_\mathcal{E}{}'$*-def* $\mathcal{S}_\mathcal{E}$*-def* $\mathcal{E}$*-from-def* **by** *blast*
then have ‹*neg* $a \in \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{E}{}')$›
using *C-in* **using** *image-iff* **by** *fastforce*
then have *a-in-SJ*: ‹$a \in \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{S}_\mathcal{J})$›
using *a-from-E-to-J* **by** *presburger*
have ‹$\exists \mathcal{C}'{\in}\mathcal{N}.\ a \in \mathit{fset}\ (\mathit{A\text{-}of}\ \mathcal{C}')$›
using *C-is C-in* **unfolding** $\mathcal{S}_\mathcal{E}{}'$*-def* $\mathcal{S}_\mathcal{E}$*-def* **by** (*smt* (*verit*, *ccfv-threshold*)
AF.sel(2) *IntE J'-is* $\mathcal{E}$*-from-def a-in-*$\mathcal{E}$ *bot-fset.rep-eq empty-iff equiv-*$\mathcal{E}$*-enabled-*$\mathcal{N}$
finsert.rep-eq insert-iff mem-Collect-eq to-V.elims to-V-neg)
then have ‹$a \in \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{N})$›
by *blast*
then have ‹$a \in \bigcup(\mathit{fset}\ `\ \mathit{A\text{-}of}\ `\ \mathcal{N}')$›

using *nj-as a-in-SJ* **unfolding** $\mathcal{N}'$*-def* **by** *simp*
then show ‹$\mathcal{C} \in \mathcal{E}$*-from* $\mathcal{N}'$›
using *C-is* **unfolding** $\mathcal{E}$*-from-def* **by** *blast*
qed
ultimately have ‹$J \models_p 2\ \mathcal{S}_\mathcal{E}'$›
unfolding $\mathcal{S}_\mathcal{E}'$*-def* $\mathcal{S}_\mathcal{E}$*-def* **using** *subset-model-p2* **by** (*metis* (*no-types*, *lifting*))
then have ‹$\neg\ J \models_p 2\ \mathcal{S}_\mathcal{J}$›
using *subset-not-model* S'*-unsat* **unfolding** $\mathcal{S}'$*-def* **by** *blast*
then have ‹$\exists\, J' \in Js.\ fset\ (A\text{-}of\ (\mathcal{J}'\text{-}of\ J')) \subseteq total\text{-}strip\ J$›
unfolding *propositional-model2-def* $\mathcal{S}_\mathcal{J}$*-def propositional-projection-def enabled-projection-def* **using** *Js-is*
by (*smt* (*verit*) *Collect-cong Set.empty-def* $\mathcal{S}_\mathcal{J}$*-def enabled-def image-iff mem-Collect-eq*)
then obtain J' **where** J'*-in*: ‹$J' \in Js$› **and** *A-of-J'-in*: ‹$fset\ (A\text{-}of\ (\mathcal{J}'\text{-}of\ J')) \subseteq total\text{-}strip\ J$›
by *blast*
then have *enab-nj'*: ‹$enabled\text{-}set\ \mathcal{N}\ J'$›
using *Js-enab* **by** *blast*
then have ‹*sound-cons.entails-neg* $(Pos\ `\ (\mathcal{M}'\text{-}of\ J'\ proj_J\ J') \cup fml\text{-}ext\ `\ (fset\ (A\text{-}of\ (\mathcal{J}'\text{-}of\ J'))))\ (Pos\ `\ F\text{-}of\ `\ \mathcal{N}'\text{-}of\ J')$›
using *fsets-from-J* **by** *auto*
moreover have ‹$(\mathcal{M}'\text{-}of\ J')\ proj_J\ J' \subseteq (\mathcal{M}'\text{-}of\ J')\ proj_J\ J$›
proof −
have ‹$\mathcal{C} \in \mathcal{M}'\text{-}of\ J' \Longrightarrow enabled\ \mathcal{C}\ J' \Longrightarrow enabled\ \mathcal{C}\ J$› **for** $\mathcal{C}$
proof −
assume *C-in*: ‹$\mathcal{C} \in \mathcal{M}'\text{-}of\ J'$› **and**
‹$enabled\ \mathcal{C}\ J'$›
then have ‹$fset\ (A\text{-}of\ \mathcal{C}) \subseteq fset\ (A\text{-}of\ (\mathcal{J}'\text{-}of\ J'))$›
using *fsets-from-J*[*OF enab-nj'*] **by** *blast*
then show ‹$enabled\ \mathcal{C}\ J$›
using *A-of-J'-in* **unfolding** *enabled-def* **by** *auto*
qed
then have ‹$(\mathcal{C} \in \mathcal{M}'\text{-}of\ J' \wedge enabled\ \mathcal{C}\ J') \Longrightarrow (\mathcal{C} \in \mathcal{M}'\text{-}of\ J' \wedge enabled\ \mathcal{C}\ J)$› **for** $\mathcal{C}$
by (*smt* (*verit*, *ccfv-threshold*))
then have ‹$\{\mathcal{C}.\ \mathcal{C} \in \mathcal{M}'\text{-}of\ J' \wedge enabled\ \mathcal{C}\ J'\} \subseteq \{\mathcal{C}.\ \mathcal{C} \in \mathcal{M}'\text{-}of\ J' \wedge enabled\ \mathcal{C}\ J\}$›
by *blast*
then show ‹$(\mathcal{M}'\text{-}of\ J')\ proj_J\ J' \subseteq (\mathcal{M}'\text{-}of\ J')\ proj_J\ J$›
unfolding *enabled-projection-def* **by** *blast*
qed
ultimately have *entails-one*: ‹*sound-cons.entails-neg* $(Pos\ `\ (\mathcal{M}'\text{-}of\ J'\ proj_J\ J) \cup fml\text{-}ext\ `\ (fset\ (A\text{-}of\ (\mathcal{J}'\text{-}of\ J'))))\ (Pos\ `\ F\text{-}of\ `\ \mathcal{N}'\text{-}of\ J')$›
using *sound-cons.entails-subsets*
by (*smt* (*verit*, *ccfv-SIG*) *Un-absorb1 Un-assoc Un-left-commute image-Un neg-ext-sound-cons-rel.entails-subsets subset-refl sup.cobounded1*)
have *subs-MJ*: ‹$Pos\ `\ (\mathcal{M}'\text{-}of\ J'\ proj_J\ J) \cup$

fml-ext ‘ (fset (A-of (𝒥′-of J′))) ⊆ Pos ‘ (ℳ′ proj_J J) ∪ fml-ext ‘ (total-strip J)›
using *J′-in A-of-J′-in* **using** *enabled-projection-def* **unfolding** *ℳ′-def* **by** *auto*
have *subs-N*: ‹*Pos ‘ F-of ‘ (𝒩′-of J′) ⊆ Pos ‘ F-of ‘ 𝒩′*›
using *J′-in* **unfolding** *𝒩′-def* **by** *blast*
show ‹*sound-cons.entails-neg (fml-ext ‘ total-strip J ∪ Pos ‘ (ℳ′ proj_J J)) (Pos ‘ F-of ‘ 𝒩′)*›
using *neg-ext-sound-cons-rel.entails-subsets*[*OF subs-MJ subs-N entails-one*]
by (*simp add*: *Un-commute*)
qed

ultimately
show ‹∃ℳ′ 𝒩′. *ℳ′ ⊆ ℳ ∧ 𝒩′ ⊆ 𝒩 ∧ finite ℳ′ ∧ finite 𝒩′ ∧ ℳ′ ⊨s_AF 𝒩′*›
by *blast*
qed
qed

interpretation *AF-sound-cons-rel*: *consequence-relation to-AF bot AF-entails-sound*
by (*rule AF-ext-sound-cons-rel*)

lemma *f-of-to-AF* [*simp*]: ‹*F-of ‘ to-AF ‘ N = N*›
unfolding *to-AF-def* **by** *force*

lemma *to-AF-proj-J* [*simp*]: ‹*to-AF ‘ M proj_J J = M*›
unfolding *to-AF-def enabled-projection-def enabled-def* **by** *force*

lemma *enabled-to-AF-set* [*simp*]: ‹*enabled-set (to-AF ‘ N) J*›
unfolding *enabled-set-def enabled-def to-AF-def* **by** *simp*

lemma *pos-not-pos-empty* [*simp*]: ‹{*to-V C* |*C*. *C ∈ Pos ‘ N ∧ ¬ is-Pos C*} = {}›
by *auto*

lemma *pos-not-pos-simp* [*simp*]:
‹{*to-V C* |*C*. *C ∈ U ∪ Pos ‘ M ∧ ¬ is-Pos C*} = {*to-V C* |*C*. *C ∈ U ∧ ¬ is-Pos C*}›
by *auto*

lemma *pos-pos-simp* [*simp*]: ‹{*to-V C* |*C*. *C ∈ Pos ‘ F-of ‘ N ∧ is-Pos C*} = *F-of ‘ N*›
by *force*

lemma *proj-F-of* [*simp*]: ‹{*C*. *F-of C ∈ M*} *proj_J J = M*›
proof (*intro equalityI subsetI*)
fix *x*
assume *x-in*: ‹*x ∈* {*C*. *F-of C ∈ M*} *proj_J J*›
define *C*::(′*f*,′*v*) *AF* **where** ‹*C = to-AF x*›
then show ‹*x ∈ M*›

using *x-in* **unfolding** *enabled-projection-def enabled-def to-AF-def* **by** *auto*
next
fix *x*
assume *x-in*: ‹$x \in M$›
define C::('*f*,'*v*) *AF* **where** ‹C = *to-AF x*›
then show ‹$x \in$ {C. *F-of C* ∈ *M*} *proj*$_J$ *J*›
using *x-in* **unfolding** *enabled-projection-def enabled-def to-AF-def*
by (*metis* (*mono-tags*, *lifting*) *AF.sel*(1) *AF.sel*(2) *bot-fset.rep-eq empty-subsetI mem-Collect-eq*)
qed

lemma *f-of-F-of* [*simp*]: ‹*F-of* ‘ {C. *F-of C* ∈ *M*} = *M*›
proof (*intro equalityI subsetI*)
fix *x*
assume *x-in*: ‹*x* ∈ *F-of* ‘ {C. *F-of C* ∈ *M*}›
define C **where** ‹C = *to-AF x*›
then show ‹*x* ∈ *M*›
using *x-in* **by** *blast*
next
fix *x*
assume *x-in*: ‹*x* ∈ *M*›
define C **where** ‹C = *to-AF x*›
then show ‹*x* ∈ *F-of* ‘ {C. *F-of C* ∈ *M*}›
using *x-in* **by** (*smt* (*z3*) *AF.sel*(1) *imageI mem-Collect-eq*)
qed

lemma *set-on-union-triple-split*: ‹{*f C* |C. $C \in M \cup N \cup g\ J \wedge l\ C\ J$} = {*f C* |$C$. $C \in M \wedge l\ C\ J$} ∪
{*f C* |C. $C \in N \wedge l\ C\ J$} ∪ {*f C* |C. $C \in g\ J \wedge l\ C\ J$}›
by *blast*

lemma *not-enabled-enabled-empty* [*simp*]:
‹{*F-of C* |C. $C \in$ {C. *F-of C* ∈ $Q' \wedge \neg$ *enabled C J*} ∧ *enabled C J*} = {}›
proof (*intro equalityI subsetI*)
fix *x*
assume *x-in*: ‹*x* ∈ {*F-of C* |C. $C \in$ {C. *F-of C* ∈ $Q' \wedge \neg$ *enabled C J*} ∧ *enabled C J*}›
then obtain C **where** ‹*F-of C* = *x*› **and** *c-in*: ‹$C \in$ {C. *F-of C* ∈ $Q' \wedge \neg$ *enabled C J*}› **and**
enab-c: ‹*enabled C J*›
by *blast*
then have ‹$\neg$ *enabled C J*› **using** *c-in* **by** *blast*
then have ‹*False*› **using** *enab-c* **by** *auto*
then show ‹*x* ∈ {}› **by** *auto*
qed *auto*

lemma *f-of-simp-enabled* [*simp*]: ‹{*F-of C* |C. *F-of C* ∈ *M* ∧ *enabled C J*} = *M*›
unfolding *enabled-def*
by (*smt* (*verit*, *best*) *AF.sel*(1) *AF.sel*(2) *bot-fset.rep-eq empty-subsetI mem-Collect-eq*

subsetI
subset-antisym)

lemma *f-of-enabled-simp* [*simp*]: ‹*F-of* ‘ {*C*. *F-of C* ∈ *M* ∧ *enabled C J*} = *M*›
proof −
have ‹*F-of* ‘ {*C*. *F-of C* ∈ *M* ∧ *enabled C J*} = {*F-of C* |*C*. *F-of C* ∈ *M* ∧ *enabled C J*}›
by *blast*
then show *?thesis* **by** *simp*
qed

lemma ‹(*to-AF* ‘ *M* ⊨$_{AF}$ *to-AF* ‘ *N*) ≡ (*M* ⊨ *N*)›
unfolding *AF-entails-def* **by** *simp*

sublocale *ext-cons-rel-std*: *consequence-relation Pos* (*to-AF bot*) *AF-cons-rel.entails-neg*
using *AF-cons-rel.ext-cons-rel* **.**

sublocale *sound-cons-rel*: *consequence-relation Pos bot sound-cons.entails-neg*
using *sound-cons.ext-cons-rel* **.**

lemma *pos-in-pos-simp* [*simp*]: ‹{*C*. *Pos C* ∈ *Pos* ‘ *N*} = *N*› **by** *auto*
lemma *neg-not-in-pos-simp* [*simp*]: ‹{*C*. *Neg C* ∈ *Pos* ‘ *N*} = {}› **by** *auto*
lemma *neg-in-pos-simp* [*simp*]: ‹{*C*. *Neg C* ∈ *P* ∨ *Neg C* ∈ *Pos* ‘ *M*} = {*C*. *Neg C* ∈ *P*}› **by** *blast*
lemma *pos-in-pos-partial-simp* [*simp*]: ‹{*C*. *Pos C* ∈ *P* ∨ *Pos C* ∈ *Pos* ‘ *M*} = {*C*. *Pos C* ∈ *P*} ∪ *M*› **by** *auto*

lemma ‹(*to-AF* ‘ *M* ⊨*s*$_{AF}$ *to-AF* ‘ *N*) ≡ (*M* ⊨*s N*)›
proof −
{
fix *M N*
assume *m-to-n*: ‹*M* ⊨*s N*›
have ‹*to-AF* ‘ *M* ⊨*s*$_{AF}$ *to-AF* ‘ *N*›
unfolding *AF-entails-sound-def sound-cons.entails-neg-def*
proof (*rule allI*, *rule impI*)
fix *J*
have ‹{*C*. *Pos C* ∈ *fml-ext* ‘ *total-strip J*} ∪ *M* ⊨*s N* ∪ {*C*. *Neg C* ∈ *fml-ext* ‘ *total-strip J*}›
proof −
have *m-in*: ‹*M* ⊆ {*to-V C* |*C*. (*C* ∈ *fml-ext* ‘ *total-strip J* ∨ *C* ∈ *Pos* ‘ *M*) ∧ *is-Pos C*}›
by *force*
show ‹ {*C*. *Pos C* ∈ *fml-ext* ‘ *total-strip J*} ∪ *M* ⊨*s*
N ∪ {*C*. *Neg C* ∈ *fml-ext* ‘ *total-strip J*}›
using *m-to-n m-in* **by** (*meson Un-subset-iff sound-cons.entails-subsets subset-refl*)
qed

```
    then show ‹{C. Pos C ∈ fml-ext ‘ total-strip J ∪ Pos ‘ (to-AF ‘ M proj_J
J)} ∪
     {C. Neg C ∈ Pos ‘ F-of ‘ to-AF ‘ N} ⊨s {C. Pos C ∈ Pos ‘ F-of ‘ to-AF ‘
N} ∪
     {C. Neg C ∈ fml-ext ‘ total-strip J ∪ Pos ‘ (to-AF ‘ M proj_J J)}›
     by simp
   qed
  } moreover {
   fix M N
   assume m-af-entails-n: ‹to-AF ‘ M ⊨s_AF to-AF ‘ N›
   have ‹M ⊆ M' ⟹ N ⊆ N' ⟹ M' ∪ N' = UNIV ⟹ M' ⊨s N'› for M' N'
   proof −
    fix M' N'
    assume m-in: ‹M ⊆ M'› and
     n-in: ‹N ⊆ N'› and
     union-mnp-is-univ: ‹M' ∪ N' = UNIV›
    {
     assume ‹M' ∩ N' ≠ {}›
     then have ‹M' ⊨s N'›
      using sound-cons.entails-reflexive sound-cons.entails-subsets
      by (meson Int-lower1 Int-lower2 sound-cons.entails-cond-reflexive)
    }
    moreover {
     assume empty-inter-mp-np: ‹M' ∩ N' = {}›
     define Jpos where ‹Jpos = {v. to-V (fml-ext v) ∈ M' ∧ is-Pos v}›
     define Jneg where ‹Jneg = {v |v. to-V (fml-ext v) ∉ M' ∧ ¬ is-Pos v}›
     have total-J-pos-neg: ‹to-V ‘ (Jpos ∪ Jneg) = UNIV›
     proof
      show ‹to-V ‘ (Jpos ∪ Jneg) ⊆ UNIV› by simp
     next
      show ‹UNIV ⊆ to-V ‘ (Jpos ∪ Jneg) ›
      proof
       fix v::'v
       define v-p where ‹v-p = Pos v›
       define v-n where ‹v-n = Neg v›
       have ‹v-p ∉ Jpos ⟹ v-n ∈ Jneg›
        unfolding v-p-def v-n-def Jpos-def Jneg-def by simp
       then show ‹v ∈ to-V ‘ (Jpos ∪ Jneg)›
        unfolding v-p-def v-n-def by force
      qed
     qed
     define Jstrip where ‹Jstrip = Jpos ∪ Jneg›
     have interp-Jstrip: ‹is-interpretation Jstrip›
      unfolding is-interpretation-def
     proof (rule ballI, rule ballI, rule impI, rule ccontr)
      fix v1 v2
      assume v1-in: ‹v1 ∈ Jstrip› and
       v2-in: ‹v2 ∈ Jstrip› and
       v12-eq: ‹to-V v1 = to-V v2› and
```

```
      contra: ‹v1 ≠ v2›
    have pos-neg-cases: ‹(v1 ∈ Jpos ∧ v2 ∈ Jneg) ∨ (v1 ∈ Jneg ∧ v2 ∈ Jpos)›
      using v1-in v2-in contra unfolding Jstrip-def Jpos-def Jneg-def
    by (smt (z3) Collect-mono-iff Collect-subset Un-def is-Neg-to-V is-Pos-to-V
v12-eq)
    then have ‹to-V (fml-ext v1) ≠ to-V (fml-ext v2)›
      using empty-inter-mp-np unfolding Jneg-def Jpos-def by auto
    then show ‹False›
      using fml-ext-preserves-val[OF v12-eq] by blast
  qed
  then obtain Jinterp where Jinterp-is: strip Jinterp = Jstrip
    by (metis Rep-propositional-interpretation-cases mem-Collect-eq)
  have ‹total Jinterp›
    unfolding total-def
  proof
    fix v
    define v-p::'v sign where v-p = Pos v
    define v-n::'v sign where v-n = Neg v
    have v-p ∈ Jpos ∨ v-n ∈ Jneg
      unfolding v-p-def v-n-def Jpos-def Jneg-def by simp
    then have v-p ∈ Jstrip ∨ v-n ∈ Jstrip
      unfolding Jstrip-def by fast
    then have ‹v-p ∈J Jinterp ∨ v-n ∈J Jinterp›
      using Jinterp-is unfolding belong-to-def by simp
    then show ‹∃ vJ. vJ ∈J Jinterp ∧ to-V vJ = v›
      using v-p-def v-n-def by auto
  qed
  then obtain Jtotal where Jtotal-is: total-strip Jtotal = Jstrip
    using Jinterp-is by (metis Rep-total-interpretation-cases mem-Collect-eq)
  have ‹enabled-set (to-AF ‘ N) Jtotal›
    unfolding enabled-set-def to-AF-def enabled-def using Jtotal-is Jstrip-def
Jneg-def
    by simp
  then have ‹fml-ext ‘ total-strip Jtotal ∪ Pos ‘ M |=s∼ Pos ‘ N›
    using m-af-entails-n unfolding AF-entails-sound-def by simp
  then have entails-m-n-jtot: ‹{C. Pos C ∈ fml-ext ‘ total-strip Jtotal} ∪ M
|=s
     N ∪ {C. Neg C ∈ fml-ext ‘ total-strip Jtotal}›
    unfolding sound-cons.entails-neg-def by simp
  have ‹{C. Pos C ∈ fml-ext ‘ total-strip Jtotal} ⊆ M'›
    unfolding Jtotal-is Jstrip-def Jpos-def Jneg-def
  by (smt (verit, ccfv-threshold) UnE fml-ext-preserves-sign imageE is-Pos.simps(1)
      mem-Collect-eq subsetI to-V.simps(1))
  then have sub-mp: ‹{C. Pos C ∈ fml-ext ‘ total-strip Jtotal} ∪ M ⊆ M'›
    using m-in by simp
  have ‹{C. Neg C ∈ fml-ext ‘ total-strip Jtotal} ⊆ N'›
    unfolding Jtotal-is Jstrip-def Jpos-def Jneg-def
  proof −
    have ‹{C. Neg C ∈ fml-ext ‘ {v. to-V (fml-ext v) ∈ M' ∧ is-Pos v}} =
```

$\{\}\rangle$
using *fml-ext-preserves-sign is-Pos.simps(1)*
by (*smt* (*verit*, *ccfv-SIG*) *empty-iff imageE is-Pos.simps*(2) *mem-Collect-eq subsetI*
subset-antisym)
moreover have ‹$\{C.\ Neg\ C \in fml\text{-}ext\ `\ \{v\ |v.\ to\text{-}V\ (fml\text{-}ext\ v) \notin M' \wedge \neg\ is\text{-}Pos\ v\}\} \subseteq N'$›
using *fml-ext-preserves-sign is-Pos.simps*(2)
proof −
have ‹$\{C.\ Neg\ C \in fml\text{-}ext\ `\ \{v\ |v.\ to\text{-}V\ (fml\text{-}ext\ v) \notin M' \wedge \neg\ is\text{-}Pos\ v\}\} =$
$\{C.\ Neg\ C \in fml\text{-}ext\ `\ \{v\ |v.\ to\text{-}V\ (fml\text{-}ext\ v) \in N' \wedge \neg\ is\text{-}Pos\ v\}\}$›
using *empty-inter-mp-np union-mnp-is-univ* **by** *auto*
also have ‹$\{C.\ Neg\ C \in fml\text{-}ext\ `\ \{v\ |v.\ to\text{-}V\ (fml\text{-}ext\ v) \in N' \wedge \neg\ is\text{-}Pos\ v\}\} \subseteq N'$›
using *fml-ext-preserves-sign is-Pos.simps*(2)
by (*smt* (*verit*, *best*) *imageE mem-Collect-eq subsetI to-V.simps*(2))

finally show ‹$\{C.\ Neg\ C \in fml\text{-}ext\ `\ \{v\ |v.\ to\text{-}V\ (fml\text{-}ext\ v) \notin M' \wedge \neg\ is\text{-}Pos\ v\}\} \subseteq N'$› **.**
qed
ultimately show ‹$\{C.\ Neg\ C \in fml\text{-}ext\ `$
$(\{v.\ to\text{-}V\ (fml\text{-}ext\ v) \in M' \wedge is\text{-}Pos\ v\} \cup$
$\{v\ |v.\ to\text{-}V\ (fml\text{-}ext\ v) \notin M' \wedge \neg\ is\text{-}Pos\ v\})\}$
$\subseteq N'$›
by *blast*
qed
then have *sub-np*: ‹$N \cup \{C.\ Neg\ C \in fml\text{-}ext\ `\ total\text{-}strip\ Jtotal\} \subseteq N'$›
using *n-in* **by** *blast*
have ‹$M' \models s\ N'$›
using *sound-cons.entails-subsets*[*OF sub-mp sub-np entails-m-n-jtot*] **.**
}
ultimately show $M' \models s\ N'$ **by** *blast*
qed
then have *supsets-entail*: ‹$\forall M'\ N'.\ (M' \supseteq M \wedge N' \supseteq N \wedge M' \cup N' = UNIV) \longrightarrow M' \models s\ N'$›
by *clarsimp*
then have ‹$M \models s\ N$›
using *sound-cons.entails-supsets* **by** *simp*
}
ultimately show ‹$(to\text{-}AF\ `\ M \models s_{AF}\ to\text{-}AF\ `\ N) \equiv (M \models s\ N)$›
by (*smt* (*verit*, *best*))
qed

lemma *strong-entails-bot-cases*: ‹$\mathcal{N} \cup \{AF.Pair\ bot\ A\} \models s_{AF}\ \{AF.Pair\ bot\ B\} \Longrightarrow$
$(\forall\ J.\ fset\ B \subseteq total\text{-}strip\ J \longrightarrow$
$(fml\text{-}ext\ `\ total\text{-}strip\ J \cup Pos\ `\ (\mathcal{N}\ proj_J\ J)) \models s_\sim\ \{Pos\ bot\} \vee fset\ A \subseteq total\text{-}strip\ J)$›

proof −
assume ‹$\mathcal{N} \cup \{AF.Pair\ bot\ A\} \models s_{AF}\ \{AF.Pair\ bot\ B\}$›
then have ‹*fset* $B \subseteq$ *total-strip* $J \Longrightarrow$
(*fml-ext* ‘ *total-strip* $J \cup$ *Pos* ‘ ($\mathcal{N}$ $proj_J$ J) $\cup$ *Pos* ‘ ({$AF.Pair\ bot\ A$} $proj_J$ J))
$\models s_\sim$ {*Pos bot*}›
for J
unfolding *AF-entails-sound-def*
by (*metis* (*no-types*, *lifting*) *AF.sel*(*1*) *AF.sel*(*2*) *distrib-proj-singleton enabled-def*
enabled-set-def image-Un image-empty image-insert singletonD sup-assoc)
then have ‹*fset* $B \subseteq$ *total-strip* $J \Longrightarrow$
((*fml-ext* ‘ *total-strip* J) $\cup$ *Pos* ‘ ($\mathcal{N}$ $proj_J$ J)) $\models s_\sim$ {*Pos bot*} $\vee$ *enabled* ($AF.Pair\ bot\ A$) J›
for J
by (*smt* (*verit*, *ccfv-SIG*) *enabled-projection-def ex-in-conv image-empty*
mem-Collect-eq singletonD sup-bot-right)
then show *?thesis*
by (*simp add*: *enabled-def*)
qed

lemma *strong-entails-bot-cases-Union*:
‹$\mathcal{N} \cup \mathcal{M} \models s_{AF}\ \{AF.Pair\ bot\ B\} \Longrightarrow (\forall\ x \in \mathcal{M}.\ F\text{-}of\ x = bot) \Longrightarrow$
($\forall\ J.$ *fset* $B \subseteq$ *total-strip* $J \longrightarrow$
(*fml-ext* ‘ *total-strip* $J \cup$ *Pos* ‘ ($\mathcal{N}$ $proj_J$ J)) $\models s_\sim$ {*Pos bot*} $\vee$
($\exists\ A \in A\text{-}of$ ‘ $\mathcal{M}.$ *fset* $A \subseteq$ *total-strip* J))›
proof −
assume $\mathcal{N}$*-union-*$\mathcal{M}$*-entails-Pair-bot-B*: ‹$\mathcal{N} \cup \mathcal{M} \models s_{AF}\ \{AF.Pair\ bot\ B\}$› **and**
‹$\forall\ x \in \mathcal{M}.\ F\text{-}of\ x = bot$›
then show *?thesis*
proof (*cases* ‹$\mathcal{M} = \{\}$›)
case *True*
then show *?thesis*
using *AF-entails-sound-def* $\mathcal{N}$*-union-*$\mathcal{M}$*-entails-Pair-bot-B enabled-def enabled-set-def*
by *auto*
next
case *False*
then show *?thesis*
proof (*intro allI impI*)
fix J
assume *B-in-J*: ‹*fset* $B \subseteq$ *total-strip* J›
then show ‹(*fml-ext* ‘ *total-strip* $J \cup$ *Pos* ‘ ($\mathcal{N}$ $proj_J$ J)) $\models s_\sim$ {*Pos bot*} $\vee$
($\exists\ A \in A\text{-}of$ ‘ $\mathcal{M}.$ *fset* $A \subseteq$ *total-strip* J)›
proof (*cases* ‹$\exists\ A \in A\text{-}of$ ‘ $\mathcal{M}.$ *fset* $A \subseteq$ *total-strip* J›)
case *True*
then show *?thesis*
by *blast*
next

case *False*
then have ‹∀ *A* ∈ *A-of* ‘ $\mathcal{M}$. ¬ *fset A* ⊆ *total-strip J*›
by *blast*
then have ‹$\mathcal{M}$ *proj*$_J$ *J* = {}›
by (*simp add*: *enabled-def enabled-projection-def*)
then show *?thesis*
using $\mathcal{N}$*-union-*$\mathcal{M}$*-entails-Pair-bot-B*[*unfolded AF-entails-sound-def*, *rule-format*]
B-in-J distrib-proj enabled-def enabled-set-def
by *fastforce*
qed
qed
qed
qed

lemma *AF-entails-sound-right-disjunctive*: ‹(∃ $\mathcal{C}'$ ∈ *A*. $\mathcal{M}$ ⊨*s*$_{AF}$ {$\mathcal{C}'$}) ⟹ $\mathcal{M}$ ⊨*s*$_{AF}$ *A*›
proof −
assume ‹∃ $\mathcal{C}'$ ∈ *A*. $\mathcal{M}$ ⊨*s*$_{AF}$ {$\mathcal{C}'$}›
then obtain $\mathcal{C}'$ **where** ‹$\mathcal{M}$ ⊨*s*$_{AF}$ {$\mathcal{C}'$}› **and**
‹$\mathcal{C}'$ ∈ *A*›
by *blast*
then show ‹$\mathcal{M}$ ⊨*s*$_{AF}$ *A*›
by (*meson AF-sound-cons-rel.entails-subsets empty-subsetI insert-subset subset-refl*)
qed

6.1 Local saturation

To fully capture completeness for splitting, we need to use weaker notions of saturation and fairness.

definition *locally-saturated* :: ‹('*f*, '*v*) *AF set* ⇒ *bool*› **where**
‹*locally-saturated* $\mathcal{N}$ ≡
to-AF bot ∈ $\mathcal{N}$ ∨
(∃ *J* :: '*v total-interpretation*. *J* ⊨*p* $\mathcal{N}$ ∧ *saturated* ($\mathcal{N}$ *proj*$_J$ *J*))›

definition *locally-fair* :: ‹('*f*, '*v*) *AF set infinite-llist* ⇒ *bool*› **where**
‹*locally-fair* $\mathcal{N}i$ ≡
(∃ *i*. *to-AF bot* ∈ *llnth* $\mathcal{N}i$ *i*)
∨ (∃ *J* :: '*v total-interpretation*. *J* ⊨*p* *lim-inf* $\mathcal{N}i$ ∧
Inf-from (*lim-inf* $\mathcal{N}i$ *proj*$_J$ *J*) ⊆ (⋃ *i*. *Red*$_I$ (*llnth* $\mathcal{N}i$ *i proj*$_J$ *J*)))›

end

locale *strong-statically-complete-annotated-calculus* =
calculus-with-annotated-consrel bot Inf entails entails-sound Red$_I$ *Red*$_F$ *fml asn*

+ *S-calculus*: *calculus to-AF bot SInf AF-entails* $SRed_I$ $SRed_F$
for
bot :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨ *50*) **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨*s 50*) **and**
Red_I :: *'f set* ⇒ *'f inference set* **and**
Red_F :: *'f set* ⇒ *'f set* **and**
fml :: ‹*'v* :: *countable* ⇒ *'f*› **and**
asn :: ‹*'f sign* ⇒ *'v sign set*› **and**
SInf :: (*'f*, *'v*) *AF inference set* **and**
$SRed_I$ :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF inference set* **and**
$SRed_F$ :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set*
+ **assumes**
strong-static-completeness: ‹*locally-saturated* $\mathcal{N}$ ⟹ $\mathcal{N} \models_{AF} \{$*to-AF bot*$\}$ ⟹ *to-AF bot* ∈ $\mathcal{N}$›

locale *strong-dynamically-complete-annotated-calculus* =
calculus-with-annotated-consrel bot Inf entails entails-sound Red_I Red_F *fml asn*
+ *S-calculus*: *calculus to-AF bot SInf AF-entails* $SRed_I$ $SRed_F$
for
bot :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨ *50*) **and**
entails-sound :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** ⊨*s 50*) **and**
Red_I :: *'f set* ⇒ *'f inference set* **and**
Red_F :: *'f set* ⇒ *'f set* **and**
fml :: ‹*'v* :: *countable* ⇒ *'f*› **and**
asn :: ‹*'f sign* ⇒ *'v sign set*› **and**
SInf :: (*'f*, *'v*) *AF inference set* **and**
$SRed_I$ :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF inference set* **and**
$SRed_F$ :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set*
+ **assumes**
strong-dynamic-completeness: ‹*is-derivation S-calculus.derive* $\mathcal{N}$ *i* ⟹ *locally-fair* $\mathcal{N}$ *i* ⟹
llhd $\mathcal{N}$ *i* $\models_{AF} \{$*to-AF bot*$\}$ ⟹ ∃ *i*. *to-AF bot* ∈ *llnth* $\mathcal{N}$ *i i*›

end

7 Lifting to Non-ground Calculi

The section 3.1 to 3.3 of the report are covered by the current section. Various forms of lifting are proven correct. These allow to obtain the dynamic refutational completeness of a non-ground calculus from the static refutational completeness of its ground counterpart.

theory *Light-Lifting-to-Non-Ground-Calculi*
imports

Saturation-Framework.Intersection-Calculus
Saturation-Framework.Calculus-Variations
Well-Quasi-Orders.Minimal-Elements
begin

7.1 Standard Lifting

locale *light-standard-lifting* = *inference-system Inf-F* +
ground: *calculus Bot-G Inf-G entails-G Red-I-G Red-F-G*
for
Inf-F :: ‹*'f inference set*› **and**
Bot-G :: ‹*'g set*› **and**
Inf-G :: ‹*'g inference set*› **and**
entails-G :: ‹*'g set* ⇒ *'g set* ⇒ *bool*› (**infix** $\models G$ *50*) **and**
Red-I-G :: ‹*'g set* ⇒ *'g inference set*› **and**
Red-F-G :: ‹*'g set* ⇒ *'g set*›
+ **fixes**
Bot-F :: ‹*'f set*› **and**
$\mathcal{G}$*-F* :: ‹*'f* ⇒ *'g set*› **and**
$\mathcal{G}$*-I* :: ‹*'f inference* ⇒ *'g inference set option*›
assumes
Bot-F-not-empty: *Bot-F* ≠ {} **and**
Bot-map-not-empty: ‹*B* ∈ *Bot-F* ⟹ $\mathcal{G}$*-F B* ≠ {}› **and**
Bot-map: ‹*B* ∈ *Bot-F* ⟹ $\mathcal{G}$*-F B* ⊆ *Bot-G*› **and**

inf-map: ‹*ι* ∈ *Inf-F* ⟹ $\mathcal{G}$*-I ι* ≠ *None* ⟹ *the* ($\mathcal{G}$*-I ι*) ⊆ *Red-I-G* ($\mathcal{G}$*-F* (*concl-of ι*))›
begin

abbreviation $\mathcal{G}$*-Fset* :: ‹*'f set* ⇒ *'g set*› **where**
‹$\mathcal{G}$*-Fset N* ≡ ⋃ ($\mathcal{G}$*-F* ‘ *N*)›

lemma $\mathcal{G}$*-subset*: ‹*N1* ⊆ *N2* ⟹ $\mathcal{G}$*-Fset N1* ⊆ $\mathcal{G}$*-Fset N2*› **by** *auto*

abbreviation *entails-*$\mathcal{G}$ :: ‹*'f set* ⇒ *'f set* ⇒ *bool*› (**infix** $\models\mathcal{G}$ *50*) **where**
‹*N1* $\models\mathcal{G}$ *N2* ≡ $\mathcal{G}$*-Fset N1* $\models G$ $\mathcal{G}$*-Fset N2*›

lemma *subs-Bot-G-entails*:
assumes
not-empty: ‹*sB* ≠ {}› **and**
in-bot: ‹*sB* ⊆ *Bot-G*›
shows ‹*sB* $\models G$ *N*›
proof −
have ‹∃ *B*. *B* ∈ *sB*› **using** *not-empty* **by** *auto*
then obtain *B* **where** *B-in*: ‹*B* ∈ *sB*› **by** *auto*
then have *r-trans*: ‹{*B*} $\models G$ *N*› **using** *ground.bot-entails-all in-bot* **by** *auto*
have *l-trans*: ‹*sB* $\models G$ {*B*}› **using** *B-in ground.subset-entailed* **by** *auto*
then show *?thesis* **using** *r-trans ground.entails-trans*[*of sB* {*B*}] **by** *auto*
qed

sublocale *consequence-relation Bot-F entails-$\mathcal{G}$*
proof
show *Bot-F* $\neq \{\}$ **using** *Bot-F-not-empty* **.**
next
show ‹*B*∈*Bot-F* $\Longrightarrow \{B\} \models_{\mathcal{G}} N$› **for** *B N*
proof −
assume ‹*B* ∈ *Bot-F*›
then show ‹$\{B\} \models_{\mathcal{G}} N$›
using *Bot-map ground.bot-entails-all*[*of* - *$\mathcal{G}$-Fset N*] *subs-Bot-G-entails Bot-map-not-empty*
by *auto*
qed
next
fix *N1 N2* :: ‹*'f set*›
assume
‹*N2* ⊆ *N1*›
then show ‹*N1* $\models_{\mathcal{G}}$ *N2*› **using** *$\mathcal{G}$-subset ground.subset-entailed* **by** *auto*
next
fix *N1 N2*
assume
N1-entails-C: ‹∀ *C* ∈ *N2*. *N1* $\models_{\mathcal{G}} \{C\}$›
show ‹*N1* $\models_{\mathcal{G}}$ *N2*› **using** *ground.all-formulas-entailed N1-entails-C*
by (*smt* (*z3*) *UN-E UN-I ground.entail-set-all-formulas singletonI*)
next
fix *N1 N2 N3*
assume
‹*N1* $\models_{\mathcal{G}}$ *N2*› **and** ‹*N2* $\models_{\mathcal{G}}$ *N3*›
then show ‹*N1* $\models_{\mathcal{G}}$ *N3*› **using** *ground.entails-trans* **by** *blast*
qed

definition *Red-I-$\mathcal{G}$* :: *'f set* ⇒ *'f inference set* **where**
‹*Red-I-$\mathcal{G}$ N* = {*ι* ∈ *Inf-F*. (*$\mathcal{G}$-I ι* ≠ *None* ∧ *the* (*$\mathcal{G}$-I ι*) ⊆ *Red-I-G* (*$\mathcal{G}$-Fset N*))
∨ (*$\mathcal{G}$-I ι* = *None* ∧ *$\mathcal{G}$-F* (*concl-of ι*) ⊆ *$\mathcal{G}$-Fset N* ∪ *Red-F-G* (*$\mathcal{G}$-Fset N*))}›

definition *Red-F-$\mathcal{G}$* :: *'f set* ⇒ *'f set* **where**
‹*Red-F-$\mathcal{G}$ N* = {*C*. ∀ *D* ∈ *$\mathcal{G}$-F C*. *D* ∈ *Red-F-G* (*$\mathcal{G}$-Fset N*)}›
end

7.2 Strong Standard Lifting

locale *light-strong-standard-lifting* = *inference-system Inf-F* +
ground: *calculus Bot-G Inf-G entails-G Red-I-G Red-F-G*
for
Inf-F :: ‹*'f inference set*› **and**
Bot-G :: ‹*'g set*› **and**
Inf-G :: ‹*'g inference set*› **and**
entails-G :: ‹*'g set* ⇒ *'g set* ⇒ *bool*› (**infix** $\models_G$ *50*) **and**
Red-I-G :: ‹*'g set* ⇒ *'g inference set*› **and**

Red-F-G :: ‹*'g set* ⇒ *'g set*›
+ **fixes**
Bot-F :: ‹*'f set*› **and**
$\mathcal{G}$*-F* :: ‹*'f* ⇒ *'g set*› **and**
$\mathcal{G}$*-I* :: ‹*'f inference* ⇒ *'g inference set option*›
assumes
Bot-F-not-empty: *Bot-F* ≠ {} **and**
Bot-map-not-empty: ‹*B* ∈ *Bot-F* ⟹ $\mathcal{G}$*-F B* ≠ {}› **and**
Bot-map: ‹*B* ∈ *Bot-F* ⟹ $\mathcal{G}$*-F B* ⊆ *Bot-G*› **and**

strong-inf-map: ‹*ι* ∈ *Inf-F* ⟹ $\mathcal{G}$*-I ι* ≠ *None* ⟹ *concl-of* ‘ (*the* ($\mathcal{G}$*-I ι*)) ⊆ ($\mathcal{G}$*-F* (*concl-of ι*))› **and**
inf-map-in-Inf: ‹*ι* ∈ *Inf-F* ⟹ $\mathcal{G}$*-I ι* ≠ *None* ⟹ *the* ($\mathcal{G}$*-I ι*) ⊆ *Inf-G*›
begin

sublocale *light-standard-lifting Inf-F Bot-G Inf-G* (⊨*G*) *Red-I-G Red-F-G Bot-F* $\mathcal{G}$*-F* $\mathcal{G}$*-I*
proof
show *Bot-F* ≠ {} **using** *Bot-F-not-empty* **.**
next
fix *B*
assume *b-in*: *B* ∈ *Bot-F*
show $\mathcal{G}$*-F B* ≠ {} **using** *Bot-map-not-empty*[*OF b-in*] **.**
next
fix *B*
assume *b-in*: *B* ∈ *Bot-F*
show $\mathcal{G}$*-F B* ⊆ *Bot-G* **using** *Bot-map*[*OF b-in*] **.**

next
fix *ι*
assume *i-in*: *ι* ∈ *Inf-F* **and**
some-g: $\mathcal{G}$*-I ι* ≠ *None*
show *the* ($\mathcal{G}$*-I ι*) ⊆ *Red-I-G* ($\mathcal{G}$*-F* (*concl-of ι*))
proof
fix *ιG*
assume *ig-in1*: *ιG* ∈ *the* ($\mathcal{G}$*-I ι*)
then have *ig-in2*: *ιG* ∈ *Inf-G* **using** *inf-map-in-Inf*[*OF i-in some-g*] **by** *blast*
show *ιG* ∈ *Red-I-G* ($\mathcal{G}$*-F* (*concl-of ι*))
using *strong-inf-map*[*OF i-in some-g*] *ground.Red-I-of-Inf-to-N*[*OF ig-in2*]
ig-in1 **by** *blast*
qed
qed

end

7.3 Lifting with a Family of Tiebreaker Orderings

locale *light-tiebreaker-lifting* =
empty-ord?: *light-standard-lifting Inf-F Bot-G Inf-G entails-G Red-I-G Red-F-G*

Bot-F $\mathcal{G}$*-F* $\mathcal{G}$*-I*
for
Bot-F :: ‹*'f set*› **and**
Inf-F :: ‹*'f inference set*› **and**
Bot-G :: ‹*'g set*› **and**
entails-G :: ‹*'g set* ⇒ *'g set* ⇒ *bool*› (**infix** ⊨*G 50*) **and**
Inf-G :: ‹*'g inference set*› **and**
Red-I-G :: ‹*'g set* ⇒ *'g inference set*› **and**
Red-F-G :: ‹*'g set* ⇒ *'g set*› **and**
$\mathcal{G}$*-F* :: *'f* ⇒ *'g set* **and**
$\mathcal{G}$*-I* :: *'f inference* ⇒ *'g inference set option*
+ **fixes**
Prec-F-g :: ‹*'g* ⇒ *'f* ⇒ *'f* ⇒ *bool*›
assumes
all-wf: *minimal-element* (*Prec-F-g g*) *UNIV*
begin

definition *Red-F-*$\mathcal{G}$ :: *'f set* ⇒ *'f set* **where**
‹*Red-F-*$\mathcal{G}$ *N* = {*C*. ∀ *D* ∈ $\mathcal{G}$*-F C*. *D* ∈ *Red-F-G* ($\mathcal{G}$*-Fset N*) ∨ (∃ *E* ∈ *N*. *Prec-F-g D E C* ∧ *D* ∈ $\mathcal{G}$*-F E*)}›

lemma *Prec-trans*:
assumes
‹*Prec-F-g D A B*› **and**
‹*Prec-F-g D B C*›
shows
‹*Prec-F-g D A C*›
using *minimal-element.po assms* **unfolding** *po-on-def transp-on-def* **by** (*smt* (*z3*) *UNIV-I all-wf*)

lemma *prop-nested-in-set*: *D* ∈ *P C* ⟹ *C* ∈ {*C*. ∀ *D* ∈ *P C*. *A D* ∨ *B C D*} ⟹ *A D* ∨ *B C D*
by *blast*

lemma *Red-F-*$\mathcal{G}$*-equiv-def*:
‹*Red-F-*$\mathcal{G}$ *N* = {*C*. ∀ *Di* ∈ $\mathcal{G}$*-F C*. *Di* ∈ *Red-F-G* ($\mathcal{G}$*-Fset N*) ∨
(∃ *E* ∈ (*N* − *Red-F-*$\mathcal{G}$ *N*). *Prec-F-g Di E C* ∧ *Di* ∈ $\mathcal{G}$*-F E*)}›
proof
show ‹*local.Red-F-*$\mathcal{G}$ *N* ⊆ {*C*. ∀ *Di*∈$\mathcal{G}$*-F C*. *Di* ∈ *Red-F-G* ($\mathcal{G}$*-Fset N*) ∨
(∃ *E*∈*N* − *local.Red-F-*$\mathcal{G}$ *N*. *Prec-F-g Di E C* ∧ *Di* ∈ $\mathcal{G}$*-F E*)}›
proof
fix *C*
assume *C-in*: ‹*C* ∈ *Red-F-*$\mathcal{G}$ *N*›
have ‹*Di* ∈ $\mathcal{G}$*-F C* ⟹ ∀ *E*∈*N* − *local.Red-F-*$\mathcal{G}$ *N*. *Prec-F-g Di E C* ⟶ *Di* ∉ $\mathcal{G}$*-F E* ⟹ *Di* ∈ *Red-F-G* ($\mathcal{G}$*-Fset N*)› **for** *Di*
proof −
fix *D*
assume *D-in*: ‹*D* ∈ $\mathcal{G}$*-F C*› **and**

not-sec-case: ‹$\forall E \in N - \mathit{Red\text{-}F\text{-}}\mathcal{G}\ N.\ \mathit{Prec\text{-}F\text{-}g}\ D\ E\ C \longrightarrow D \notin \mathcal{G}\text{-}F\ E$›
have *C-in-unfolded*: $C \in \{C.\ \forall Di \in \mathcal{G}\text{-}F\ C.\ Di \in \mathit{Red\text{-}F\text{-}G}\ (\mathcal{G}\text{-}\mathit{Fset}\ N) \vee (\exists E{\in}N.\ \mathit{Prec\text{-}F\text{-}g}\ Di\ E\ C \wedge Di \in \mathcal{G}\text{-}F\ E)\}$
using *C-in* **unfolding** *Red-F-G-def* **.**
have *neg-not-sec-case*: ‹$\neg\ (\exists E{\in}N - \mathit{Red\text{-}F\text{-}}\mathcal{G}\ N.\ \mathit{Prec\text{-}F\text{-}g}\ D\ E\ C \wedge D \in \mathcal{G}\text{-}F\ E)$›
using *not-sec-case* **by** *clarsimp*
have *unfol-C-D*: ‹$D \in \mathit{Red\text{-}F\text{-}G}\ (\mathcal{G}\text{-}\mathit{Fset}\ N) \vee (\exists E{\in}N.\ \mathit{Prec\text{-}F\text{-}g}\ D\ E\ C \wedge D \in \mathcal{G}\text{-}F\ E)$›
using *prop-nested-in-set*[*of D* $\mathcal{G}$*-F C* $\lambda x.\ x \in \mathit{Red\text{-}F\text{-}G}\ (\bigcup\ (\mathcal{G}\text{-}F\ `\ N))$ $\lambda x\ y.\ \exists E \in N.\ \mathit{Prec\text{-}F\text{-}g}\ y\ E\ x \wedge y \in \mathcal{G}\text{-}F\ E$, *OF D-in C-in-unfolded*] **by** *blast*
show ‹$D \in \mathit{Red\text{-}F\text{-}G}\ (\mathcal{G}\text{-}\mathit{Fset}\ N)$›
proof (*rule ccontr*)
assume *contrad*: ‹$D \notin \mathit{Red\text{-}F\text{-}G}\ (\mathcal{G}\text{-}\mathit{Fset}\ N)$›
have *non-empty*: ‹$\exists E{\in}N.\ \mathit{Prec\text{-}F\text{-}g}\ D\ E\ C \wedge D \in \mathcal{G}\text{-}F\ E$› **using** *contrad unfol-C-D* **by** *auto*
define *B* **where** ‹$B = \{E \in N.\ \mathit{Prec\text{-}F\text{-}g}\ D\ E\ C \wedge D \in \mathcal{G}\text{-}F\ E\}$›
then have *B-non-empty*: ‹$B \neq \{\}$› **using** *non-empty* **by** *auto*
interpret *minimal-element Prec-F-g D UNIV* **using** *all-wf*[*of D*] **.**
obtain $F :: 'f$ **where** *F*: ‹$F = \mathit{min\text{-}elt}\ B$› **by** *auto*
then have *D-in-F*: ‹$D \in \mathcal{G}\text{-}F\ F$› **unfolding** *B-def* **using** *non-empty*
by (*smt* (*z3*) *Sup-UNIV Sup-upper UNIV-I contra-subsetD empty-iff empty-subsetI mem-Collect-eq min-elt-mem unfol-C-D*)
have *F-prec*: ‹*Prec-F-g D F C*› **using** *F min-elt-mem*[*of B*, *OF - B-non-empty*] **unfolding** *B-def* **by** *auto*
have *F-not-in*: ‹$F \notin \mathit{Red\text{-}F\text{-}}\mathcal{G}\ N$›
proof
assume *F-in*: ‹$F \in \mathit{Red\text{-}F\text{-}}\mathcal{G}\ N$›
have *unfol-F-D*: ‹$D \in \mathit{Red\text{-}F\text{-}G}\ (\mathcal{G}\text{-}\mathit{Fset}\ N) \vee (\exists G{\in}N.\ \mathit{Prec\text{-}F\text{-}g}\ D\ G\ F \wedge D \in \mathcal{G}\text{-}F\ G)$›
using *F-in D-in-F* **unfolding** *Red-F-G-def* **by** *auto*
then have ‹$\exists G{\in}N.\ \mathit{Prec\text{-}F\text{-}g}\ D\ G\ F \wedge D \in \mathcal{G}\text{-}F\ G$› **using** *contrad D-in* **unfolding** *Red-F-G-def* **by** *auto*
then obtain *G* **where** *G-in*: ‹$G \in N$› **and** *G-prec*: ‹*Prec-F-g D G F*› **and** *G-map*: ‹$D \in \mathcal{G}\text{-}F\ G$› **by** *auto*
have ‹*Prec-F-g D G C*› **using** *G-prec F-prec Prec-trans* **by** *blast*
then have ‹$G \in B$› **unfolding** *B-def* **using** *G-in G-map* **by** *auto*
then show ‹*False*› **using** *F G-prec min-elt-minimal*[*of B G*, *OF - B-non-empty*] **by** *auto*
qed
have ‹$F \in N$› **using** *F* **by** (*metis B-def B-non-empty mem-Collect-eq min-elt-mem top-greatest*)
then have ‹$F \in N - \mathit{Red\text{-}F\text{-}}\mathcal{G}\ N$› **using** *F-not-in* **by** *auto*
then show ‹*False*›
using *D-in-F neg-not-sec-case F-prec* **by** *blast*
qed
qed

then show ‹$C \in \{C.\ \forall Di \in \mathcal{G}\text{-}F\ C.\ Di \in Red\text{-}F\text{-}G\ (\mathcal{G}\text{-}Fset\ N) \vee (\exists E \in N - local.Red\text{-}F\text{-}\mathcal{G}\ N.\ Prec\text{-}F\text{-}g\ Di\ E\ C \wedge Di \in \mathcal{G}\text{-}F\ E)\}$›
by *clarsimp*
qed
next
show ‹$\{C.\ \forall Di \in \mathcal{G}\text{-}F\ C.\ Di \in Red\text{-}F\text{-}G\ (\mathcal{G}\text{-}Fset\ N) \vee (\exists E \in N - local.Red\text{-}F\text{-}\mathcal{G}\ N.\ Prec\text{-}F\text{-}g\ Di\ E\ C \wedge Di \in \mathcal{G}\text{-}F\ E)\} \subseteq local.Red\text{-}F\text{-}\mathcal{G}\ N$›
proof
fix C
assume ‹$C \in \{C.\ \forall Di \in \mathcal{G}\text{-}F\ C.\ Di \in Red\text{-}F\text{-}G\ (\mathcal{G}\text{-}Fset\ N) \vee (\exists E \in N - local.Red\text{-}F\text{-}\mathcal{G}\ N.\ Prec\text{-}F\text{-}g\ Di\ E\ C \wedge Di \in \mathcal{G}\text{-}F\ E)\}$›
then have *only-if*: ‹$\forall D \in \mathcal{G}\text{-}F\ C.\ D \in Red\text{-}F\text{-}G\ (\mathcal{G}\text{-}Fset\ N) \vee (\exists E \in N - Red\text{-}F\text{-}\mathcal{G}\ N.\ Prec\text{-}F\text{-}g\ D\ E\ C \wedge D \in \mathcal{G}\text{-}F\ E)$›
by *clarsimp*
then show ‹$C \in Red\text{-}F\text{-}\mathcal{G}\ N$›
unfolding *Red-F-G-def* **by** *auto*
qed
qed

lemma *not-red-map-in-map-not-red*: ‹$\mathcal{G}\text{-}Fset\ N - Red\text{-}F\text{-}G\ (\mathcal{G}\text{-}Fset\ N) \subseteq \mathcal{G}\text{-}Fset\ (N - Red\text{-}F\text{-}\mathcal{G}\ N)$›
proof
fix D
assume
D-hyp: ‹$D \in \mathcal{G}\text{-}Fset\ N - Red\text{-}F\text{-}G\ (\mathcal{G}\text{-}Fset\ N)$›
interpret *minimal-element Prec-F-g D UNIV* **using** *all-wf*[*of D*] **.**
have *D-in*: ‹$D \in \mathcal{G}\text{-}Fset\ N$› **using** *D-hyp* **by** *blast*
have *D-not-in*: ‹$D \notin Red\text{-}F\text{-}G\ (\mathcal{G}\text{-}Fset\ N)$› **using** *D-hyp* **by** *blast*
have *exist-C*: ‹$\exists C.\ C \in N \wedge D \in \mathcal{G}\text{-}F\ C$› **using** *D-in* **by** *auto*
define B **where** ‹$B = \{C \in N.\ D \in \mathcal{G}\text{-}F\ C\}$›
obtain C **where** C: ‹$C = min\text{-}elt\ B$› **by** *auto*
have *C-in-N*: ‹$C \in N$›
using *exist-C* **by** (*metis B-def C empty-iff mem-Collect-eq min-elt-mem top-greatest*)
have *D-in-C*: ‹$D \in \mathcal{G}\text{-}F\ C$›
using *exist-C* **by** (*metis B-def C empty-iff mem-Collect-eq min-elt-mem top-greatest*)
have *C-not-in*: ‹$C \notin Red\text{-}F\text{-}\mathcal{G}\ N$›
proof
assume *C-in*: ‹$C \in Red\text{-}F\text{-}\mathcal{G}\ N$›
have ‹$D \in Red\text{-}F\text{-}G\ (\mathcal{G}\text{-}Fset\ N) \vee (\exists E \in N.\ Prec\text{-}F\text{-}g\ D\ E\ C \wedge D \in \mathcal{G}\text{-}F\ E)$›
using *C-in D-in-C* **unfolding** *Red-F-G-def* **by** *auto*
then show ‹*False*›
proof
assume ‹$D \in Red\text{-}F\text{-}G\ (\mathcal{G}\text{-}Fset\ N)$›
then show ‹*False*› **using** *D-not-in* **by** *simp*
next
assume ‹$\exists E \in N.\ Prec\text{-}F\text{-}g\ D\ E\ C \wedge D \in \mathcal{G}\text{-}F\ E$›
then show ‹*False*›

using *C* **by** (*metis* (*no-types*, *lifting*) *B-def UNIV-I empty-iff mem-Collect-eq min-elt-minimal top-greatest*)
qed
qed
show ‹*D* ∈ *G-Fset* (*N* − *Red-F-G N*)› **using** *D-in-C C-not-in C-in-N* **by** *blast*
qed

lemma *Red-F-Bot-F*: ‹*B* ∈ *Bot-F* ⟹ *N* ⊨*G* {*B*} ⟹ *N* − *Red-F-G N* ⊨*G* {*B*}›
proof −
fix *B N*
assume
B-in: ‹*B* ∈ *Bot-F*› **and**
N-entails: ‹*N* ⊨*G* {*B*}›
then have *to-bot*: ‹*G-Fset N* − *Red-F-G* (*G-Fset N*) ⊨*G G-F B›
using *ground.Red-F-Bot Bot-map*
by (*smt* (*z3*) *cSup-singleton ground.entail-set-all-formulas image-insert image-is-empty subsetCE*)
have *from-f*: ‹*G-Fset* (*N* − *Red-F-G N*) ⊨*G G-Fset N* − *Red-F-G* (*G-Fset N*)›
using *ground.subset-entailed*[*OF not-red-map-in-map-not-red*] **by** *blast*
then have ‹*G-Fset* (*N* − *Red-F-G N*) ⊨*G G-F B*› **using** *to-bot ground.entails-trans* **by** *blast*
then show ‹*N* − *Red-F-G N* ⊨*G* {*B*}› **using** *Bot-map* **by** *simp*
qed

lemma *Red-F-of-subset-F*: ‹*N* ⊆ *N'* ⟹ *Red-F-G N* ⊆ *Red-F-G N'*›
using *ground.Red-F-of-subset* **unfolding** *Red-F-G-def*
by (*smt* (*verit*, *ccfv-threshold*) *Collect-mono G-subset subset-iff*)

lemma *Red-I-of-subset-F*: ‹*N* ⊆ *N'* ⟹ *Red-I-G N* ⊆ *Red-I-G N'*›
using *Collect-mono G-subset subset-iff ground.Red-I-of-subset* **unfolding** *Red-I-G-def*
by (*smt* (*z3*) *ground.Red-F-of-subset Un-iff*)

lemma *Red-F-of-Red-F-subset-F*: ‹*N'* ⊆ *Red-F-G N* ⟹ *Red-F-G N* ⊆ *Red-F-G* (*N* − *N'*)›
proof
fix *N N' C*
assume
N'-in-Red-F-N: ‹*N'* ⊆ *Red-F-G N*› **and**
C-in-red-F-N: ‹*C* ∈ *Red-F-G N*›
have *lem8*: ‹∀ *D* ∈ *G-F C*. *D* ∈ *Red-F-G* (*G-Fset N*) ∨ (∃ *E* ∈ (*N* − *Red-F-G N*). *Prec-F-g D E C* ∧ *D* ∈ *G-F E*)›
using *Red-F-G-equiv-def C-in-red-F-N* **by** *blast*
show ‹*C* ∈ *Red-F-G* (*N* − *N'*)› **unfolding** *Red-F-G-def*
proof (*rule*,*rule*)
fix *D*

assume ‹$D \in \mathcal{G}\text{-}F\ C$›
then have ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset N*) $\vee$ ($\exists E \in (N - $ *Red-F-*$\mathcal{G}$ *N*). *Prec-F-g D E C* $\wedge$ $D \in \mathcal{G}\text{-}F\ E$)›
using *lem8* **by** *auto*
then show ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* ($N - N'$)) $\vee$ ($\exists E \in N - N'$. *Prec-F-g D E C* $\wedge$ $D \in \mathcal{G}\text{-}F\ E$)›
proof
assume ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset N*)›
then have ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset N* $-$ *Red-F-G* ($\mathcal{G}$*-Fset N*))›
using *ground.Red-F-of-Red-F-subset*[*of Red-F-G* ($\mathcal{G}$*-Fset N*) $\mathcal{G}$*-Fset N*] **by** *auto*
then have ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* ($N -$ *Red-F-*$\mathcal{G}$ *N*))›
using *ground.Red-F-of-subset*[*OF not-red-map-in-map-not-red*[*of N*]] **by** *auto*
then have ‹$D \in$ *Red-F-G* ($\mathcal{G}$*-Fset* ($N - N'$))›
using N'*-in-Red-F-N* $\mathcal{G}$*-subset*[*of N* $-$ *Red-F-*$\mathcal{G}$ *N N* $-$ N']
by (*smt* (*z3*) *DiffE DiffI ground.Red-F-of-subset subsetCE subsetI*)
then show *?thesis* **by** *blast*
next
assume ‹$\exists E \in N -$ *Red-F-*$\mathcal{G}$ *N*. *Prec-F-g D E C* $\wedge$ $D \in \mathcal{G}\text{-}F\ E$›
then obtain *E* **where**
E-in: ‹$E \in N -$ *Red-F-*$\mathcal{G}$ *N*› **and**
E-prec-C: ‹*Prec-F-g D E C*› **and**
D-in: ‹$D \in \mathcal{G}\text{-}F\ E$›
by *auto*
have ‹$E \in N - N'$› **using** *E-in* N'*-in-Red-F-N* **by** *blast*
then show *?thesis* **using** *E-prec-C D-in* **by** *blast*
qed
qed
qed

lemma *Red-I-of-Red-F-subset-F*: ‹$N' \subseteq$ *Red-F-*$\mathcal{G}$ *N* $\Longrightarrow$ *Red-I-*$\mathcal{G}$ *N* $\subseteq$ *Red-I-*$\mathcal{G}$ ($N - N'$) ›
proof
fix *N* N' ι
assume
N'*-in-Red-F-N*: ‹$N' \subseteq$ *Red-F-*$\mathcal{G}$ *N*› **and**
i-in-Red-I-N: ‹$\iota \in$ *Red-I-*$\mathcal{G}$ *N*›
have *i-in*: ‹$\iota \in$ *Inf-F*› **using** *i-in-Red-I-N* **unfolding** *Red-I-*$\mathcal{G}$*-def* **by** *blast*
{
assume *not-none*: $\mathcal{G}\text{-}I\ \iota \neq$ *None*
have ‹$\forall \iota' \in$ *the* ($\mathcal{G}\text{-}I\ \iota$). $\iota' \in$ *Red-I-G* ($\mathcal{G}$*-Fset N*)›
using *not-none i-in-Red-I-N* **unfolding** *Red-I-*$\mathcal{G}$*-def* **by** *auto*
then have ‹$\forall \iota' \in$ *the* ($\mathcal{G}\text{-}I\ \iota$). $\iota' \in$ *Red-I-G* ($\mathcal{G}$*-Fset N* $-$ *Red-F-G* ($\mathcal{G}$*-Fset N*))›
using *not-none ground.Red-I-of-Red-F-subset* **by** *blast*
then have *ip-in-Red-I-G*: ‹$\forall \iota' \in$ *the* ($\mathcal{G}\text{-}I\ \iota$). $\iota' \in$ *Red-I-G* ($\mathcal{G}$*-Fset* ($N -$ *Red-F-*$\mathcal{G}$ *N*))›
using *not-none ground.Red-I-of-subset*[*OF not-red-map-in-map-not-red*[*of N*]] **by** *auto*

```
    then have not-none-in: ‹∀ι'∈the (𝒢-I ι). ι'∈Red-I-G (𝒢-Fset (N − N'))›
      using not-none N'-in-Red-F-N
      by (meson Diff-mono ground.Red-I-of-subset 𝒢-subset subset-iff subset-refl)
    then have the (𝒢-I ι) ⊆ Red-I-G (𝒢-Fset (N − N')) by blast
  }
  moreover {
    assume none: 𝒢-I ι = None
    have ground-concl-subs: 𝒢-F (concl-of ι) ⊆ (𝒢-Fset N ∪ Red-F-G (𝒢-Fset N))
      using none i-in-Red-I-N unfolding Red-I-𝒢-def by blast
    then have d-in-imp12: D∈𝒢-F (concl-of ι) ⟹ D∈𝒢-Fset N − Red-F-G
(𝒢-Fset N) ∨ D∈Red-F-G (𝒢-Fset N)
      by blast
    have d-in-imp1: D∈𝒢-Fset N − Red-F-G (𝒢-Fset N) ⟹ D∈𝒢-Fset (N −
N')
      using not-red-map-in-map-not-red N'-in-Red-F-N by blast
    have d-in-imp-d-in: D∈Red-F-G (𝒢-Fset N) ⟹ D∈Red-F-G (𝒢-Fset N −
Red-F-G (𝒢-Fset N))
      using ground.Red-F-of-Red-F-subset[of Red-F-G (𝒢-Fset N) 𝒢-Fset N] by
blast
    have g-subs1: 𝒢-Fset N − Red-F-G (𝒢-Fset N) ⊆ 𝒢-Fset (N − Red-F-𝒢 N)
      using not-red-map-in-map-not-red unfolding Red-F-𝒢-def by auto
    have g-subs2: 𝒢-Fset (N − Red-F-𝒢 N) ⊆ 𝒢-Fset (N − N')
      using N'-in-Red-F-N by blast
    have d-in-imp2: D∈Red-F-G (𝒢-Fset N) ⟹ D∈Red-F-G (𝒢-Fset (N −
N'))
      using ground.Red-F-of-subset ground.Red-F-of-subset[OF g-subs1]
        ground.Red-F-of-subset[OF g-subs2] d-in-imp-d-in by blast
    have 𝒢-F (concl-of ι) ⊆ (𝒢-Fset (N − N') ∪ Red-F-G (𝒢-Fset (N − N')))
      using d-in-imp12 d-in-imp1 d-in-imp2
    by (smt (z3) ground.Red-F-of-Red-F-subset ground.Red-F-of-subset UnCI UnE
Un-Diff-cancel2
      ground-concl-subs g-subs1 g-subs2 subset-iff)
  }
  ultimately show ‹ι∈Red-I-𝒢 (N − N')› using i-in unfolding Red-I-𝒢-def by
auto
qed


lemma Red-I-of-Inf-to-N-F:
  assumes
    i-in: ‹ι∈Inf-F› and
    concl-i-in: ‹concl-of ι∈N›
  shows
    ‹ι∈Red-I-𝒢 N ›
proof −
  have ‹ι∈Inf-F ⟹ 𝒢-I ι≠None ⟹ the (𝒢-I ι) ⊆ Red-I-G (𝒢-F (concl-of
ι))› using inf-map by simp
  moreover have ‹Red-I-G (𝒢-F (concl-of ι)) ⊆ Red-I-G (𝒢-Fset N)›
    using concl-i-in ground.Red-I-of-subset by blast
```

moreover have $\iota \in$ *Inf-F* $\Longrightarrow$ $\mathcal{G}$*-I* ι = *None* $\Longrightarrow$ *concl-of* $\iota \in N \Longrightarrow \mathcal{G}$*-F* (*concl-of* ι) $\subseteq \mathcal{G}$*-Fset N*
by *blast*
ultimately show *?thesis* **using** *i-in concl-i-in* **unfolding** *Red-I-*$\mathcal{G}$*-def* **by** *auto*
qed

sublocale *calculus Bot-F Inf-F entails-*$\mathcal{G}$ *Red-I-*$\mathcal{G}$ *Red-F-*$\mathcal{G}$
proof
fix *B N N'* ι
show ‹*Red-I-*$\mathcal{G}$ *N* $\subseteq$ *Inf-F*› **unfolding** *Red-I-*$\mathcal{G}$*-def* **by** *blast*
show ‹*B* $\in$ *Bot-F* $\Longrightarrow$ *N* $\models_{\mathcal{G}}$ {*B*} $\Longrightarrow$ *N* − *Red-F-*$\mathcal{G}$ *N* $\models_{\mathcal{G}}$ {*B*}› **using** *Red-F-Bot-F* **by** *simp*
show ‹*N* $\subseteq$ *N'* $\Longrightarrow$ *Red-F-*$\mathcal{G}$ *N* $\subseteq$ *Red-F-*$\mathcal{G}$ *N'*› **using** *Red-F-of-subset-F* **by** *simp*
show ‹*N* $\subseteq$ *N'* $\Longrightarrow$ *Red-I-*$\mathcal{G}$ *N* $\subseteq$ *Red-I-*$\mathcal{G}$ *N'*› **using** *Red-I-of-subset-F* **by** *simp*
show ‹*N'* $\subseteq$ *Red-F-*$\mathcal{G}$ *N* $\Longrightarrow$ *Red-F-*$\mathcal{G}$ *N* $\subseteq$ *Red-F-*$\mathcal{G}$ (*N* − *N'*)› **using** *Red-F-of-Red-F-subset-F* **by** *simp*
show ‹*N'* $\subseteq$ *Red-F-*$\mathcal{G}$ *N* $\Longrightarrow$ *Red-I-*$\mathcal{G}$ *N* $\subseteq$ *Red-I-*$\mathcal{G}$ (*N* − *N'*)› **using** *Red-I-of-Red-F-subset-F* **by** *simp*
show ‹$\iota \in$ *Inf-F* $\Longrightarrow$ *concl-of* $\iota \in N \Longrightarrow \iota \in$ *Red-I-*$\mathcal{G}$ *N*› **using** *Red-I-of-Inf-to-N-F* **by** *simp*
qed

end

lemma *wf-empty-rel*: *minimal-element* (λ- -. *False*) *UNIV*
by (*simp add*: *minimal-element.intro po-on-def transp-onI wfp-on-imp-irreflp-on*)

lemma *light-standard-empty-tiebreaker-equiv*: *light-standard-lifting Inf-F Bot-G Inf-G entails-G Red-I-G*
Red-F-G Bot-F $\mathcal{G}$*-F* $\mathcal{G}$*-I* = *light-tiebreaker-lifting Bot-F Inf-F Bot-G entails-G Inf-G Red-I-G*
Red-F-G $\mathcal{G}$*-F* $\mathcal{G}$*-I* (λ*g C C'*. *False*)
proof −
have *light-tiebreaker-lifting-axioms* (λ*g C C'*. *False*)
unfolding *light-tiebreaker-lifting-axioms-def* **using** *wf-empty-rel* **by** *simp*
then show *?thesis*
unfolding *light-standard-lifting-def light-tiebreaker-lifting-def* **by** *blast*
qed

context *light-standard-lifting*
begin

interpretation *empt-ord*: *light-tiebreaker-lifting Bot-F Inf-F Bot-G entails-G Inf-G Red-I-G*
Red-F-G $\mathcal{G}$*-F* $\mathcal{G}$*-I* λ*g C C'*. *False*
using *light-standard-empty-tiebreaker-equiv* **using** *light-standard-lifting-axioms*
by *blast*

lemma *red-f-equiv*: *empt-ord.Red-F-$\mathcal{G}$* = *Red-F-$\mathcal{G}$*
unfolding *Red-F-$\mathcal{G}$-def empt-ord.Red-F-$\mathcal{G}$-def* **by** *simp*

sublocale *calc?*: *calculus Bot-F Inf-F entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ Red-F-$\mathcal{G}$*
using *empt-ord.calculus-axioms red-f-equiv* **by** *fastforce*

lemma *grounded-inf-in-ground-inf*: $\iota \in$ *Inf-F* $\Longrightarrow$ *$\mathcal{G}$-I* $\iota \neq$ *None* $\Longrightarrow$ *the* (*$\mathcal{G}$-I* ι) $\subseteq$ *Inf-G*
using *inf-map ground.Red-I-to-Inf* **by** *blast*

abbreviation *ground-Inf-overapproximated* :: *'f set* $\Rightarrow$ *bool* **where**
ground-Inf-overapproximated N $\equiv$ *ground.Inf-from* (*$\mathcal{G}$-Fset N*)
$\subseteq$ {ι. $\exists \iota' \in$ *Inf-from N*. *$\mathcal{G}$-I* $\iota' \neq$ *None* $\wedge$ $\iota \in$ *the* (*$\mathcal{G}$-I* ι')} $\cup$ *Red-I-G* (*$\mathcal{G}$-Fset N*)

lemma *sat-inf-imp-ground-red*:
assumes
saturated N **and**
$\iota' \in$ *Inf-from N* **and**
$\mathcal{G}$-I $\iota' \neq$ *None* $\wedge$ $\iota \in$ *the* (*$\mathcal{G}$-I* ι')
shows $\iota \in$ *Red-I-G* (*$\mathcal{G}$-Fset N*)
using *assms Red-I-$\mathcal{G}$-def* **unfolding** *saturated-def* **by** *auto*

lemma *sat-imp-ground-sat*:
saturated N $\Longrightarrow$ *ground-Inf-overapproximated N* $\Longrightarrow$ *ground.saturated* (*$\mathcal{G}$-Fset N*)
unfolding *ground.saturated-def* **using** *sat-inf-imp-ground-red* **by** *auto*

end

context *light-tiebreaker-lifting*
begin

lemma *saturated-empty-order-equiv-saturated*:
saturated N = *calc.saturated N*
by (*rule refl*)

lemma *static-empty-order-equiv-static*:
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ Red-F-$\mathcal{G}$ =
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ empty-ord.Red-F-$\mathcal{G}$
unfolding *statically-complete-calculus-def*

by (*rule iffI*) (*standard*,(*standard*)[],*simp*)+

theorem *static-to-dynamic*:
statically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ empty-ord.Red-F-$\mathcal{G}$ = dynamically-complete-calculus Bot-F Inf-F entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ Red-F-$\mathcal{G}$
using *dyn-equiv-stat static-empty-order-equiv-static*
by *blast*

end

7.4 Lifting with a Family of Redundancy Criteria

locale *light-lifting-intersection = inference-system Inf-F +*
ground: inference-system-family Q Inf-G-q +
ground: consequence-relation-family Bot-G Q entails-q
for
Inf-F :: 'f inference set **and**
Bot-G :: 'g set **and**
Q :: 'q set **and**
Inf-G-q :: ‹'q ⇒ 'g inference set› **and**
entails-q :: 'q ⇒ 'g set ⇒ 'g set ⇒ bool **and**
Red-I-q :: 'q ⇒ 'g set ⇒ 'g inference set **and**
Red-F-q :: 'q ⇒ 'g set ⇒ 'g set
+ fixes
Bot-F :: 'f set **and**
$\mathcal{G}$-F-q :: 'q ⇒ 'f ⇒ 'g set **and**
$\mathcal{G}$-I-q :: 'q ⇒ 'f inference ⇒ 'g inference set option **and**
Prec-F-g :: 'g ⇒ 'f ⇒ 'f ⇒ bool
assumes
light-standard-lifting-family:
∀ q ∈ Q. light-tiebreaker-lifting Bot-F Inf-F Bot-G (entails-q q) (Inf-G-q q) (Red-I-q q) (Red-F-q q) ($\mathcal{G}$-F-q q) ($\mathcal{G}$-I-q q) Prec-F-g
begin

abbreviation *$\mathcal{G}$-Fset-q :: 'q ⇒ 'f set ⇒ 'g set* **where**
$\mathcal{G}$-Fset-q q N ≡ ⋃ ($\mathcal{G}$-F-q q ‘ N)

definition *Red-I-$\mathcal{G}$-q :: 'q ⇒ 'f set ⇒ 'f inference set* **where**
Red-I-$\mathcal{G}$-q q N = {ι ∈ Inf-F. ($\mathcal{G}$-I-q q ι ≠ None ∧ the ($\mathcal{G}$-I-q q ι) ⊆ Red-I-q q ($\mathcal{G}$-Fset-q q N))
∨ ($\mathcal{G}$-I-q q ι = None ∧ $\mathcal{G}$-F-q q (concl-of ι) ⊆ ($\mathcal{G}$-Fset-q q N ∪ Red-F-q q ($\mathcal{G}$-Fset-q q N)))}

definition *Red-F-$\mathcal{G}$-empty-q :: 'q ⇒ 'f set ⇒ 'f set* **where**
Red-F-$\mathcal{G}$-empty-q q N = {C. ∀ D ∈ $\mathcal{G}$-F-q q C. D ∈ Red-F-q q ($\mathcal{G}$-Fset-q q N)}

definition *Red-F-$\mathcal{G}$-q :: 'q ⇒ 'f set ⇒ 'f set* **where**

Red-F-$\mathcal{G}$-q q N =
{*C*. ∀ *D* ∈ *$\mathcal{G}$-F-q q C*. *D* ∈ *Red-F-q q* (*$\mathcal{G}$-Fset-q q N*) ∨ (∃ *E* ∈ *N*. *Prec-F-g D E C* ∧ *D* ∈ *$\mathcal{G}$-F-q q E*)}

abbreviation *entails-$\mathcal{G}$-q* :: *'q* ⇒ *'f set* ⇒ *'f set* ⇒ *bool* **where**
entails-$\mathcal{G}$-q q N1 N2 ≡ *entails-q q* (*$\mathcal{G}$-Fset-q q N1*) (*$\mathcal{G}$-Fset-q q N2*)

lemma *red-crit-lifting-family*:
assumes *q-in*: *q* ∈ *Q*
shows *calculus Bot-F Inf-F* (*entails-$\mathcal{G}$-q q*) (*Red-I-$\mathcal{G}$-q q*) (*Red-F-$\mathcal{G}$-q q*)
proof −
interpret *wf-lift*:
light-tiebreaker-lifting Bot-F Inf-F Bot-G entails-q q Inf-G-q q Red-I-q q
Red-F-q q $\mathcal{G}$-F-q q $\mathcal{G}$-I-q q Prec-F-g
using *light-standard-lifting-family q-in* **by** *metis*
have *Red-I-$\mathcal{G}$-q q* = *wf-lift.Red-I-$\mathcal{G}$*
unfolding *Red-I-$\mathcal{G}$-q-def wf-lift.Red-I-$\mathcal{G}$-def* **by** *blast*
moreover have *Red-F-$\mathcal{G}$-q q* = *wf-lift.Red-F-$\mathcal{G}$*
unfolding *Red-F-$\mathcal{G}$-q-def wf-lift.Red-F-$\mathcal{G}$-def* **by** *blast*
ultimately show *?thesis*
using *wf-lift.calculus-axioms* **by** *simp*
qed

lemma *red-crit-lifting-family-empty-ord*:
assumes *q-in*: *q* ∈ *Q*
shows *calculus Bot-F Inf-F* (*entails-$\mathcal{G}$-q q*) (*Red-I-$\mathcal{G}$-q q*) (*Red-F-$\mathcal{G}$-empty-q q*)
proof −
interpret *wf-lift*:
light-tiebreaker-lifting Bot-F Inf-F Bot-G entails-q q Inf-G-q q Red-I-q q
Red-F-q q $\mathcal{G}$-F-q q $\mathcal{G}$-I-q q Prec-F-g
using *light-standard-lifting-family q-in* **by** *metis*
have *Red-I-$\mathcal{G}$-q q* = *wf-lift.Red-I-$\mathcal{G}$*
unfolding *Red-I-$\mathcal{G}$-q-def wf-lift.Red-I-$\mathcal{G}$-def* **by** *blast*
moreover have *Red-F-$\mathcal{G}$-empty-q q* = *wf-lift.empty-ord.Red-F-$\mathcal{G}$*
unfolding *Red-F-$\mathcal{G}$-empty-q-def wf-lift.empty-ord.Red-F-$\mathcal{G}$-def* **by** *blast*
ultimately show *?thesis*
using *wf-lift.calc.calculus-axioms* **by** *simp*
qed

sublocale *consequence-relation-family Bot-F Q entails-$\mathcal{G}$-q*
proof (*unfold-locales*; (*intro ballI*) *?*)
show *Q* ≠ {}
by (*rule ground.Q-nonempty*)
next
fix *qi*
assume *qi-in*: *qi* ∈ *Q*

interpret *lift*: *light-tiebreaker-lifting Bot-F Inf-F Bot-G entails-q qi Inf-G-q qi*
Red-I-q qi Red-F-q qi $\mathcal{G}$-F-q qi $\mathcal{G}$-I-q qi Prec-F-g

using *qi-in* **by** (*metis light-standard-lifting-family*)

show *consequence-relation Bot-F* (*entails-$\mathcal{G}$-q qi*)
by *unfold-locales*
qed

sublocale *intersection-calculus Bot-F Inf-F Q entails-$\mathcal{G}$-q Red-I-$\mathcal{G}$-q Red-F-$\mathcal{G}$-q*
by *unfold-locales* (*auto simp*: *Q-nonempty red-crit-lifting-family*)

abbreviation *entails-$\mathcal{G}$* :: *'f set* ⇒ *'f set* ⇒ *bool* (**infix** $\models\cap\mathcal{G}$ *50*) **where**
$(\models\cap\mathcal{G}) \equiv$ *entails*

abbreviation *Red-I-$\mathcal{G}$* :: *'f set* ⇒ *'f inference set* **where**
Red-I-$\mathcal{G}$ ≡ *Red-I*

abbreviation *Red-F-$\mathcal{G}$* :: *'f set* ⇒ *'f set* **where**
Red-F-$\mathcal{G}$ ≡ *Red-F*

lemmas *entails-$\mathcal{G}$-def* = *entails-def*
lemmas *Red-I-$\mathcal{G}$-def* = *Red-I-def*
lemmas *Red-F-$\mathcal{G}$-def* = *Red-F-def*

sublocale *empty-ord*: *intersection-calculus Bot-F Inf-F Q entails-$\mathcal{G}$-q Red-I-$\mathcal{G}$-q Red-F-$\mathcal{G}$-empty-q*
by *unfold-locales* (*auto simp*: *Q-nonempty red-crit-lifting-family-empty-ord*)

abbreviation *Red-F-$\mathcal{G}$-empty* :: *'f set* ⇒ *'f set* **where**
Red-F-$\mathcal{G}$-empty ≡ *empty-ord.Red-F*

lemmas *Red-F-$\mathcal{G}$-empty-def* = *empty-ord.Red-F-def*

lemma *sat-inf-imp-ground-red-fam-inter*:
assumes
sat-n: *saturated N* **and**
i'-in: $\iota' \in$ *Inf-from N* **and**
q-in: $q \in Q$ **and**
grounding: *$\mathcal{G}$-I-q q ι'* ≠ *None* ∧ $\iota \in$ *the* (*$\mathcal{G}$-I-q q ι'*)
shows $\iota \in$ *Red-I-q q* (*$\mathcal{G}$-Fset-q q N*)
proof −
have $\iota' \in$ *Red-I-$\mathcal{G}$-q q N*
using *sat-n i'-in q-in all-red-crit calculus.saturated-def sat-int-to-sat-q*
by *blast*
then have *the* (*$\mathcal{G}$-I-q q ι'*) ⊆ *Red-I-q q* (*$\mathcal{G}$-Fset-q q N*)
by (*simp add*: *Red-I-$\mathcal{G}$-q-def grounding*)
then show *?thesis*
using *grounding* **by** *blast*
qed

abbreviation *ground-Inf-overapproximated* :: *'q* ⇒ *'f set* ⇒ *bool* **where**

ground-Inf-overapproximated q N ≡
ground.Inf-from-q q (*G*-*Fset-q q N*)
⊆ {*ι*. ∃*ι*′∈ *Inf-from N*. *G*-*I-q q ι*′ ≠ *None* ∧ *ι* ∈ *the* (*G*-*I-q q ι*′)} ∪ *Red-I-q q* (*G*-*Fset-q q N*)

abbreviation *ground-saturated* :: *′q* ⇒ *′f set* ⇒ *bool* **where**
ground-saturated q N ≡ *ground.Inf-from-q q* (*G*-*Fset-q q N*) ⊆ *Red-I-q q* (*G*-*Fset-q q N*)

lemma *sat-imp-ground-sat-fam-inter*:
saturated N ⟹ *q* ∈ *Q* ⟹ *ground-Inf-overapproximated q N* ⟹ *ground-saturated q N*
using *sat-inf-imp-ground-red-fam-inter* **by** *auto*

lemma *sat-eq-sat-empty-order*: *saturated N* = *empty-ord.saturated N*
by (*rule refl*)

lemma *static-empty-ord-inter-equiv-static-inter*:
statically-complete-calculus Bot-F Inf-F entails Red-I Red-F =
statically-complete-calculus Bot-F Inf-F entails Red-I Red-F-G-empty
unfolding *statically-complete-calculus-def*
by (*simp add*: *empty-ord.calculus-axioms calculus-axioms*)

theorem *stat-eq-dyn-ref-comp-fam-inter*: *statically-complete-calculus Bot-F Inf-F entails Red-I Red-F-G-empty* =
dynamically-complete-calculus Bot-F Inf-F entails Red-I Red-F
using *dyn-equiv-stat static-empty-ord-inter-equiv-static-inter*
by *blast*

end

end
theory *List-Extra*
imports *Main HOL−Library.Quotient-List*
begin

This theory contains some extra lemmas that were useful in proving some lemmas in `Modular_Splitting_Calculus.thy` and `Lightweight_Avatar.thy`.

lemma *map2-first-is-first* [*simp*]: ‹*length x* = *length y* ⟹ *map2* (λ *x y*. *x*) *x y* = *x*›
by (*metis fst-def map-eq-conv map-fst-zip*)

lemma *map2-second-is-second* [*simp*]: ‹*length A* = *length B* ⟹ *map2* (λ *x y*. *y*)

A B = B›
by (*metis map-eq-conv map-snd-zip snd-def*)

lemma *list-all-exists-is-exists-list-all2*:
assumes ‹*list-all* (λ *x*. ∃ *y*. *P x y*) *xs*›
shows ‹∃ *ys*. *list-all2 P xs ys*›
using *assms*
by (*induct xs*, *auto*)

lemma *ball-set-f-to-ball-set-map*: ‹(∀ *x* ∈ *set A*. *P* (*f x*)) ⟷ (∀ *x* ∈ *set* (*map f A*). *P x*)›
by *simp*

lemma *list-all-ex-to-ex-list-all2*:
‹*list-all* (λ *x*. ∃ *y*. *P x y*) *A* ⟷ (∃ *ys*. *length A* = *length ys* ∧ *list-all2* (λ *x y*. *P x y*) *A ys*)›
by (*metis list-all2-conv-all-nth list-all-exists-is-exists-list-all2 list-all-length*)

lemma *list-all2-to-map*:
assumes *lengths-eq*: ‹*length A* = *length B*›
shows ‹*list-all2* (λ *x y*. *P* (*f x y*)) *A B* ⟷ *list-all P* (*map2 f A B*)›
proof −
have ‹*list-all2* (λ *x y*. *P* (*f x y*)) *A B* ⟷ *list-all* (λ (*x*, *y*). *P* (*f x y*)) (*zip A B*)›
by (*simp add*: *lengths-eq list-all2-iff list-all-iff*)
also have ‹... ⟷ *list-all* (λ *x*. *P x*) (*map2 f A B*)›
by (*simp add*: *case-prod-beta list-all-iff*)
finally show *?thesis* **.**
qed

end

theory *Modular-Splitting-Calculus*
imports
Calculi-And-Annotations
Light-Lifting-to-Non-Ground-Calculi
List-Extra
FSet-Extra
begin

8 Core splitting calculus

In this section, we formalize an abstract version of a splitting calculus. We start by considering only two basic rules:

- Base performs an inference from our inference system;
- Unsat replaces a set of prosopositionally unsatisfiable formulas with ⊥.

locale *annotated-calculus* = *calculus-with-annotated-consrel bot FInf entails entails-sound* $FRed_I$
  $FRed_F$ *fml asn*
  for *bot* :: *'f* **and**
    FInf :: ‹*'f inference set*› **and**
    entails :: ‹[*'f set*, *'f set*] ⇒ *bool*› (**infix** ‹⊨› *50*) **and**
    entails-sound :: ‹[*'f set*, *'f set*] ⇒ *bool*› (**infix** ‹⊨*s*› *50*) **and**
    $FRed_I$ :: ‹*'f set* ⇒ *'f inference set*› **and**
    $FRed_F$ :: ‹*'f set* ⇒ *'f set*› **and**
    fml :: ‹*'v* :: *countable* ⇒ *'f*› **and**
    asn :: ‹*'f sign* ⇒ *'v sign set*›
  + **assumes**

    entails-nontrivial: ‹¬ {} ⊨ {}› **and**

    reducedness: ‹*Inf-between UNIV* ($FRed_F$ *N*) ⊆ $FRed_I$ *N*› **and**

    complete: ‹*bot* ∉ $FRed_F$ *N*› **and**

    all-red-to-bot: ‹$\mathcal{C}$ ≠ *bot* ⟹ $\mathcal{C}$ ∈ $FRed_F$ {*bot*}›
begin

notation *sound-cons.entails-neg* (**infix** ‹⊨$s_\sim$› *50*)

8.1 The inference rules

Every inference rule X is defined using two functions: X_pre and X_inf. X_inf is the inference rule itself, while X_pre are side-conditions for the rule to be applicable.

abbreviation *base-pre* :: ‹(*'f*, *'v*) *AF list* ⇒ *'f* ⇒ *bool*› **where**
  ‹*base-pre* $\mathcal{N}$ *D* ≡ *Infer* (*map F-of* $\mathcal{N}$) *D* ∈ *FInf*›

abbreviation *base-inf* :: ‹(*'f*, *'v*) *AF list* ⇒ *'f* ⇒ (*'f*, *'v*) *AF inference*› **where**
  ‹*base-inf* $\mathcal{N}$ *D* ≡ *Infer* $\mathcal{N}$ (*Pair D* (*ffUnion* (*fset-of-list* (*map A-of* $\mathcal{N}$))))›

abbreviation *unsat-pre* :: ‹(*'f*, *'v*) *AF list* ⇒ *bool*› **where**
  ‹*unsat-pre* $\mathcal{N}$ ≡ (∀ *x* ∈ *set* $\mathcal{N}$. *F-of x* = *bot*) ∧ *propositionally-unsatisfiable* (*set* $\mathcal{N}$)›

abbreviation *unsat-inf* :: ‹(*'f*, *'v*) *AF list* ⇒ (*'f*, *'v*) *AF inference*› **where**
  ‹*unsat-inf* $\mathcal{N}$ ≡ *Infer* $\mathcal{N}$ (*to-AF bot*)›

We consider first only the inference rules Base and Unsat. The optional inference and simplification rules are handled separately in the locales *splitting-calculus-extensions* and *splitting-calculus-with-simps* respectively.

inductive-set *SInf* :: ‹(*'f*, *'v*) *AF inference set*› **where**
  base: ‹*base-pre* $\mathcal{N}$ *D* ⟹ *base-inf* $\mathcal{N}$ *D* ∈ *SInf*›
| *unsat*: ‹*unsat-pre* $\mathcal{N}$ ⟹ *unsat-inf* $\mathcal{N}$ ∈ *SInf*›

The predicates in *Splitting-rules* form a valid inference system.

interpretation *SInf-inf-system*: *inference-system SInf* **.**

lemma *not-empty-entails-bot*: ‹$\neg\{\} \models \{bot\}$›
  using *entails-bot-to-entails-empty entails-nontrivial*
  by *blast*

The proof for Lemma 13 is split into two parts, for each inclusion in the set equality.

lemma *SInf-commutes-Inf1*:
  ‹*bot* $\notin \mathcal{N}$ *proj*$_J$ *J* $\Longrightarrow$ (*inference-system.Inf-from SInf* $\mathcal{N}$) ι*proj*$_J$ *J* $\subseteq$ *Inf-from* ($\mathcal{N}$ *proj*$_J$ *J*)›
proof (*intro subsetI*)
  fix ι_S
  assume *bot-not-in-proj*: ‹*bot* $\notin \mathcal{N}$ *proj*$_J$ *J*› **and**
      ι_S*-is-inf*: ‹$\iota_S \in$ (*inference-system.Inf-from SInf* $\mathcal{N}$) ι*proj*$_J$ *J*›

  have *no-enabled-prop-clause-in-*$\mathcal{N}$: ‹$\forall\ \mathcal{C} \in \mathcal{N}$. *enabled* $\mathcal{C}$ *J* $\longrightarrow$ *F-of* $\mathcal{C} \neq$ *bot*›
    using *bot-not-in-proj*
    unfolding *enabled-projection-def*
    by *blast*

  obtain ι_F **where** ι_S*-is*: ‹$\iota_S = \iota$*F-of* ι_F› **and**
        ι_F*-is-inf*: ‹$\iota_F \in$ *inference-system.Inf-from SInf* $\mathcal{N}$› **and**
        ι_F*-is-enabled*: ‹*enabled-inf* ι_F *J*›
    using ι_S*-is-inf enabled-projection-Inf-def*
    by *auto*
  then have ι_F*-in-S*: ‹$\iota_F \in$ *SInf*›
    by (*simp add*: *inference-system.Inf-from-def*)
  moreover have *prems-of-*ι_F*-subset-*$\mathcal{N}$: ‹*set* (*prems-of* ι_F) $\subseteq \mathcal{N}$›
    using ι_F*-is-inf*
    by (*simp add*: *inference-system.Inf-from-def*)
  moreover have ‹ι*F-of* $\iota_F \in$ *FInf*›
    unfolding ι*F-of-def*
    using ι_F*-in-S*
  proof (*cases* ι_F *rule*: *SInf.cases*)
    case (*base* $\mathcal{N}$ *D*)
    then show ‹*base-pre* (*prems-of* ι_F) (*F-of* (*concl-of* ι_F))›
      by *auto*
  next
    case (*unsat* $\mathcal{N}$)
    moreover have ‹*enabled-inf* ι_F *J*›
      using ι_F*-is-enabled*
      by *fastforce*
    then have ‹$\forall\ \mathcal{C} \in$ *set* $\mathcal{N}$. *F-of* $\mathcal{C} \neq$ *bot*›
    by (*metis enabled-inf-def inference.sel*(*1*) *local.unsat*(*1*) *no-enabled-prop-clause-in-*$\mathcal{N}$
        prems-of-ι_F*-subset-*$\mathcal{N}$ *subset-eq*)
    then have ‹*False*›
      using *not-empty-entails-bot unsat*(*2*) *enabled-projection-def prop-proj-in*

propositional-model-def propositionally-unsatisfiable-def **by** *auto*
ultimately show ‹*base-pre* (*prems-of* ι_F) (*F-of* (*concl-of* ι_F))›
by *blast*
qed
moreover have ‹*set* (*prems-of* (*ιF-of* ι_F)) ⊆ $\mathcal{N}$ $proj_J$ J›
using ι_F*-is-enabled prems-of-*ι_F*-subset-*$\mathcal{N}$
by (*auto simp add*: *enabled-inf-def enabled-projection-def ιF-of-def*)
ultimately have ‹*ιF-of* ι_F ∈ *Inf-from* ($\mathcal{N}$ $proj_J$ J)›
by (*simp add*: *Inf-from-def*)
then show ‹ι_S ∈ *Inf-from* ($\mathcal{N}$ $proj_J$ J)›
by (*simp add*: ι_S*-is*)
qed

lemma *SInf-commutes-Inf2*:
‹*bot* ∉ $\mathcal{N}$ $proj_J$ J ⟹ *Inf-from* ($\mathcal{N}$ $proj_J$ J) ⊆ (*inference-system.Inf-from SInf* $\mathcal{N}$) *ιproj*$_J$ J›
proof (*intro subsetI*)
fix ι_F
assume *bot-not-in-proj*: ‹*bot* ∉ $\mathcal{N}$ $proj_J$ J› **and**
ι_F*-in-inf*: ‹ι_F ∈ *Inf-from* ($\mathcal{N}$ $proj_J$ J)›

have ι_F*-is-Inf*: ‹ι_F ∈ *FInf*›
using *Inf-if-Inf-from* ι_F*-in-inf*
by *blast*
have ‹*set* (*prems-of* ι_F) ⊆ $\mathcal{N}$ $proj_J$ J›
using *Inf-from-def* ι_F*-in-inf*
by *blast*
then have ‹∀ $\mathcal{C}$ ∈ *set* (*prems-of* ι_F). ∃ A. *Pair* $\mathcal{C}$ A ∈ $\mathcal{N}$ ∧ *enabled* (*Pair* $\mathcal{C}$ A) J›
by (*smt* (*verit*, *ccfv-SIG*) *AF.collapse enabled-projection-def mem-Collect-eq subsetD*)
then have ‹*list-all* (λ $\mathcal{C}$. ∃ A. *Pair* $\mathcal{C}$ A ∈ $\mathcal{N}$ ∧ *enabled* (*Pair* $\mathcal{C}$ A) J) (*prems-of* ι_F)›
using *Ball-set*
by *blast*
then obtain *As* **where**
length-As-is-length-prems-ι_F: ‹*length* (*prems-of* ι_F) = *length As*› **and**
As-def: ‹∀ (C, A) ∈ *set* (*zip* (*prems-of* ι_F) *As*). *Pair* C A ∈ $\mathcal{N}$ ∧ *enabled* (*Pair* C A) J›
by (*smt* (*verit*, *del-insts*) *Ball-set-list-all list-all2-iff list-all-exists-is-exists-list-all2*)
define ι_S **where**
‹ι_S ≡ *Infer* [*Pair* $\mathcal{C}$ A. ($\mathcal{C}$, A) ← *zip* (*prems-of* ι_F) *As*]
(*Pair* (*concl-of* ι_F) (*ffUnion* (*fset-of-list As*)))›
have ι_F*-is-Inf2*: ‹*Infer* (*map F-of* [*Pair* $\mathcal{C}$ A. ($\mathcal{C}$, A) ← *zip* (*prems-of* ι_F) *As*]) (*concl-of* ι_F) ∈ *FInf*›
using *length-As-is-length-prems-*ι_F
by (*auto simp add*: ι_F*-is-Inf*)
then have ι_S*-is-SInf*: ‹ι_S ∈ *SInf*›

using *SInf.base*[*OF* ι_F*-is-Inf2*] *length-As-is-length-prems-*ι_F
unfolding ι_S*-def*
by *auto*
moreover have ‹*set* (*prems-of* ι_S) $\subseteq \mathcal{N}$›
unfolding ι_S*-def*
using *As-def*
by *auto*
then have ‹$\iota_S \in$ *inference-system.Inf-from SInf* $\mathcal{N}$›
using *inference-system.Inf-from-def* ι_S*-is-SInf*
by *blast*
moreover have ‹ι*F-of* $\iota_S = \iota_F$›
unfolding ι_S*-def* ι*F-of-def*
by (*simp add*: *length-As-is-length-prems-*ι_F)
moreover have ‹*enabled-inf* ι_S *J*›
unfolding *enabled-inf-def* ι_S*-def*
using *As-def*
by *auto*
ultimately have ‹$\exists\ \iota'.\ \iota_F = \iota$*F-of* $\iota' \wedge \iota' \in$ *inference-system.Inf-from SInf* $\mathcal{N}$ $\wedge$ *enabled-inf* ι' *J*›
by *blast*
then show ‹$\iota_F \in$ (*inference-system.Inf-from SInf* $\mathcal{N}$) ι*proj*$_J$ *J*›
unfolding *enabled-projection-Inf-def*
by *simp*
qed

We use *bot* $\notin$ *?*$\mathcal{N}$ *proj*$_J$ *?J* $\Longrightarrow$ *SInf-inf-system.Inf-from ?*$\mathcal{N}$ ι*proj*$_J$ *?J* $\subseteq$ *Inf-from* (*?*$\mathcal{N}$ *proj*$_J$ *?J*) and *bot* $\notin$ *?*$\mathcal{N}$ *proj*$_J$ *?J* $\Longrightarrow$ *Inf-from* (*?*$\mathcal{N}$ *proj*$_J$ *?J*) $\subseteq$ *SInf-inf-system.Inf-from ?*$\mathcal{N}$ ι*proj*$_J$ *?J* to put the Lemma 13 together into a single proof.

lemma *SInf-commutes-Inf*:
‹*bot* $\notin \mathcal{N}$ *proj*$_J$ *J* $\Longrightarrow$ (*inference-system.Inf-from SInf* $\mathcal{N}$) ι*proj*$_J$ *J* = *Inf-from* ($\mathcal{N}$ *proj*$_J$ *J*)›
using *SInf-commutes-Inf1 SInf-commutes-Inf2*
by *blast*

theorem *SInf-sound-wrt-entails-sound*: ‹$\iota_S \in$ *SInf* $\Longrightarrow$ *set* (*prems-of* ι_S) $\models s_{AF}$ {*concl-of* ι_S}›
proof −
assume ‹$\iota_S \in$ *SInf*›
then show ‹*set* (*prems-of* ι_S) $\models s_{AF}$ {*concl-of* ι_S}›
proof (*cases* ι_S *rule*: *SInf.cases*)
case (*base* $\mathcal{N}$ *D*)
assume ‹*base-pre* $\mathcal{N}$ *D*›
then have *inf-is-sound*: ‹*set* (*map F-of* $\mathcal{N}$) $\models s$ {*D*}›
using *sound*
by *fastforce*
then show *?thesis*
unfolding *AF-entails-sound-def sound-cons.entails-neg-def*

```
    proof (intro allI impI)
      fix J
      assume ‹enabled-set {concl-of ι_S} J›
      then have Pair-D-A-of-𝒩-is-enabled: ‹enabled-set {concl-of ι_S} J›
        using base
        by simp
      then have ‹F-of ‘ {concl-of ι_S} = {D}›
        using base
        by simp
      moreover have ‹fset (ffUnion (fset-of-list (map A-of 𝒩))) ⊆ total-strip J›
        using Pair-D-A-of-𝒩-is-enabled
        unfolding enabled-set-def enabled-def
        by (simp add: local.base(1))
      then have ‹fBall (fset-of-list (map A-of 𝒩)) (λ As. fset As ⊆ total-strip J)›
        using fset-ffUnion-subset-iff-all-fsets-subset
        by fast
      then have ‹∀ As ∈ set (map A-of 𝒩). fset As ⊆ total-strip J›
        by (meson fset-of-list-elem)
      then have ‹∀ 𝒞 ∈ set 𝒩. enabled 𝒞 J›
        unfolding enabled-def
        by simp
      then have ‹set 𝒩 proj_J J = F-of ‘ set 𝒩›
        unfolding enabled-projection-def
        by auto
      moreover have ‹{C. Pos C ∈ fml-ext ‘ total-strip J ∪ Pos ‘ F-of ‘ set 𝒩}
⊨s {D}›
        using inf-is-sound
       by (smt (verit, ccfv-threshold) UnCI imageI list.set-map mem-Collect-eq
           sound-cons.entails-subsets subsetI)
      moreover have ‹{C. Neg C ∈ Pos ‘ F-of ‘ {concl-of ι_S}} = {}›
        by fast
     ultimately show ‹{C. Pos C ∈ fml-ext ‘ total-strip J ∪ Pos ‘ (set (prems-of
ι_S) proj_J J)} ∪
                      {C. Neg C ∈ Pos ‘ F-of ‘ {concl-of ι_S}} ⊨s
                      {C. Pos C ∈ Pos ‘ F-of ‘ {concl-of ι_S}} ∪
                       {C. Neg C ∈ fml-ext ‘ total-strip J ∪ Pos ‘ (set (prems-of ι_S)
proj_J J)}›
        using base
          by (smt (verit, del-insts) UnCI imageI inference.sel(1) inference.sel(2)
mem-Collect-eq
            sound-cons.entails-subsets subsetI)
    qed
  next
    case (unsat 𝒩)
    assume pre-cond: ‹unsat-pre 𝒩›
    then have heads-of-𝒩-are-bot: ‹∀ x ∈ set 𝒩. F-of x = bot› and
          𝒩-is-prop-unsat: ‹propositionally-unsatisfiable (set 𝒩)›
      by blast+
    then have ‹proj_⊥ (set 𝒩) = set 𝒩›
```

using *heads-of-$\mathcal{N}$-are-bot propositional-projection-def*
by *blast*
then have ‹$\forall$ *J*. *bot* $\in$ (*set* $\mathcal{N}$) *proj*$_J$ *J*›
using *$\mathcal{N}$-is-prop-unsat propositional-model-def propositionally-unsatisfiable-def*
by *force*
then show *?thesis*
unfolding *AF-entails-sound-def sound-cons.entails-neg-def*
using *unsat*
by *auto*
(*smt* (*verit*) *Un-insert-right insertI1 insert-absorb sound-cons.bot-entails-empty sound-cons.entails-subsets subsetI sup-bot-right*)
qed
qed

The lifted calculus provides a consequence relation and a sound inference system.

interpretation *AF-sound-cons-rel*: *consequence-relation* ‹*to-AF bot*› ‹($\models s_{AF}$)›
by (*rule AF-ext-sound-cons-rel*)

interpretation *SInf-sound-inf-system*: *sound-inference-system SInf* ‹*to-AF bot*› ‹($\models s_{AF}$)›
by (*standard*, *auto simp add*: *SInf-sound-wrt-entails-sound*)

8.2 The redundancy criterion

definition *SRed*$_F$ :: ‹('*f*, '*v*) *AF set* $\Rightarrow$ ('*f*, '*v*) *AF set*› **where**
‹*SRed*$_F$ $\mathcal{N}$ = { *AF.Pair C A* | *C A*. $\forall$ $\mathcal{J}$. *total-strip* $\mathcal{J}$ $\supseteq$ *fset A* $\longrightarrow$ *C* $\in$ *FRed*$_F$ ($\mathcal{N}$ *proj*$_J$ $\mathcal{J}$) }
$\cup$ { *AF.Pair C A* | *C A*. $\exists$ $\mathcal{C}$ $\in$ $\mathcal{N}$. *F-of* $\mathcal{C}$ = *C* $\wedge$ *A-of* $\mathcal{C}$ |$\subset$| *A* }›

definition *SRed*$_I$ :: ‹('*f*, '*v*) *AF set* $\Rightarrow$ ('*f*, '*v*) *AF inference set*› **where**
‹*SRed*$_I$ $\mathcal{N}$ = { *base-inf* $\mathcal{M}$ $\mathcal{C}$ | $\mathcal{M}$ $\mathcal{C}$. *base-pre* $\mathcal{M}$ $\mathcal{C}$ $\wedge$
($\forall$ $\mathcal{J}$. { *base-inf* $\mathcal{M}$ $\mathcal{C}$ } ι*proj*$_J$ $\mathcal{J}$ $\subseteq$ *FRed*$_I$ ($\mathcal{N}$ *proj*$_J$ $\mathcal{J}$)) }
$\cup$ { *unsat-inf* $\mathcal{M}$ | $\mathcal{M}$. *unsat-pre* $\mathcal{M}$ $\wedge$ *to-AF bot* $\in$ $\mathcal{N}$ }›

lemma *sredI-$\mathcal{N}$-proj-J-subset-redI-proj-J*: ‹*to-AF bot* $\notin$ $\mathcal{N}$ $\Longrightarrow$ (*SRed*$_I$ $\mathcal{N}$) ι*proj*$_J$ *J* $\subseteq$ *FRed*$_I$ ($\mathcal{N}$ *proj*$_J$ *J*)›
proof −
assume ‹*to-AF bot* $\notin$ $\mathcal{N}$›
then have *SRed*$_I$*-$\mathcal{N}$-is*:
‹*SRed*$_I$ $\mathcal{N}$ = { *base-inf* $\mathcal{M}$ $\mathcal{C}$ | $\mathcal{M}$ $\mathcal{C}$. *base-pre* $\mathcal{M}$ $\mathcal{C}$ $\wedge$
($\forall$ $\mathcal{J}$. {*base-inf* $\mathcal{M}$ $\mathcal{C}$} ι*proj*$_J$ $\mathcal{J}$ $\subseteq$ *FRed*$_I$ ($\mathcal{N}$ *proj*$_J$ $\mathcal{J}$)) }›
using *SRed*$_I$*-def*
by *auto*
then show ‹(*SRed*$_I$ $\mathcal{N}$) ι*proj*$_J$ *J* $\subseteq$ *FRed*$_I$ ($\mathcal{N}$ *proj*$_J$ *J*)›
proof (*cases* ‹(*SRed*$_I$ $\mathcal{N}$) ι*proj*$_J$ *J* = {}›)
case *True*
then show *?thesis*

```
    by fast
  next
    case False
    then obtain ι_S where ‹ι_S ∈ SRed_I 𝒩›
      using enabled-projection-Inf-def
      by fastforce
    then have ‹{ι_S} ιproj_J J ⊆ FRed_I (𝒩 proj_J J)›
      using SRed_I-𝒩-is
      by auto
    then show ?thesis
      using SRed_I-𝒩-is enabled-projection-Inf-def
      by force
  qed
qed


lemma bot-not-in-sredF-𝒩: ‹to-AF bot ∉ SRed_F 𝒩›
proof −
  have ‹to-AF bot ∉ { AF.Pair C A | C A. ∀ 𝒥. total-strip 𝒥 ⊇ fset A ⟶ C ∈
FRed_F (𝒩 proj_J 𝒥) }›
    by (simp add: complete to-AF-def)
  moreover have ‹to-AF bot ∉ { AF.Pair C A | C A. ∃ 𝒞 ∈ 𝒩. F-of 𝒞 = C ∧
A-of 𝒞 |⊂| A }›
    by (simp add: to-AF-def)
  moreover have ‹to-AF bot ∉ { AF.Pair C A | C A. ∀ J. ¬ fset A ⊆ total-strip
J }›
    by (simp add: to-AF-def)
  ultimately show ?thesis
    using SRed_F-def
    by auto
qed
```

We need to set things up for the proof of lemma 18. We first restrict $SRed_I$ to BASE inferences (under the name $ARed_I$) and show that it is a redundancy criterion. And then we consider the case of UNSAT inferences separately.

```
definition ARed_F :: ‹('f, 'v) AF set ⇒ ('f, 'v) AF set› where
  ‹ARed_F 𝒩 ≡ SRed_F 𝒩›

definition ARed_I :: ‹('f, 'v) AF set ⇒ ('f, 'v) AF inference set› where
  ‹ARed_I 𝒩 ≡ { base-inf ℳ 𝒞 | ℳ 𝒞. base-pre ℳ 𝒞 ∧
                (∀ 𝒥. { base-inf ℳ 𝒞 } ιproj_J 𝒥 ⊆ FRed_I (𝒩 proj_J 𝒥)) }›

definition AInf :: ‹('f, 'v) AF inference set› where
  ‹AInf ≡ { base-inf 𝒩 D | 𝒩 D. base-pre 𝒩 D }›

definition 𝒢_F :: ‹'v total-interpretation ⇒ ('f, 'v) AF ⇒ 'f set› where
  ‹𝒢_F 𝒥 𝒞 ≡ {𝒞} proj_J 𝒥›
```

definition $\mathcal{G}_I$ :: ‹'v *total-interpretation* ⇒ ('f, 'v) *AF inference* ⇒ 'f *inference set*›
where
‹$\mathcal{G}_I$ $\mathcal{J}$ ι ≡ {ι} $\iota proj_J$ $\mathcal{J}$›

We define a wellfounded ordering on A-formulas to strengthen $ARed_I$. Basically, $A \leftarrow \mathcal{C} \sqsupset A \leftarrow \mathcal{C}'$ if $\mathcal{C} \subset \mathcal{C}'$.

definition *tiebreaker-order* :: ‹('f, 'v :: *countable*) *AF rel*› **where**
‹*tiebreaker-order* ≡ { ($\mathcal{C}$, $\mathcal{C}'$). *F-of* $\mathcal{C}$ = *F-of* $\mathcal{C}'$ ∧ *A-of* $\mathcal{C}$ |⊂| *A-of* $\mathcal{C}'$ }›

abbreviation *sqsupset-is-tiebreaker-order* (**infix** ‹⊐› *50*) **where**
‹$\mathcal{C}$ ⊐ $\mathcal{C}'$ ≡ ($\mathcal{C}$, $\mathcal{C}'$) ∈ *tiebreaker-order*›

lemma *tiebreaker-order-is-strict-partial-order*: ‹*po-on* (⊐) *UNIV*›
unfolding *po-on-def irreflp-on-def transp-on-def tiebreaker-order-def*
by *auto*

lemma *wfp-on-fsubset*: ‹*wfp-on* (|⊂|) *UNIV*›
using *wf-fsubset*
by *auto*

lemma *wfp-on-tiebreaker-order*: ‹*wfp-on* (⊐) (*UNIV* :: ('f, 'v) *AF set*)›
unfolding *wfp-on-def*
proof (*intro notI*)
assume ‹∃ *f*. ∀ *i*. *f i* ∈ *UNIV* ∧ *f* (*Suc i*) ⊐ *f i*›
then obtain *f* **where** *f-is*: ‹∀ *i*. *f i* ∈ *UNIV* ∧ *f* (*Suc i*) ⊐ *f i*›
by *auto*
define *f'* **where** ‹*f'* = (λ *i*. *A-of* (*f i*))›

have ‹∀ *i*. *f'* *i* ∈ *UNIV* ∧ *f'* (*Suc i*) |⊂| *f'* *i*›
using *f-is*
unfolding *f'-def tiebreaker-order-def*
by *auto*
then show ‹*False*›
using *wfp-on-fsubset*
unfolding *wfp-on-def*
by *blast*
qed

We can lift inferences from $FRed_I$ to $ARed_I$.

interpretation *lift-from-FRed-to-ARed*: *light-tiebreaker-lifting* ‹{*to-AF bot*}› *AInf* ‹{*bot*}› ‹(⊨∩)›
FInf $FRed_I$ $FRed_F$ ‹$\mathcal{G}_F$ $\mathcal{J}$› ‹*Some* ∘ $\mathcal{G}_I$ $\mathcal{J}$› ‹λ-. (⊐)›
proof (*standard*)
fix *N*
show ‹$FRed_I$ *N* ⊆ *FInf*›
using *Red-I-to-Inf*
by *presburger*
next
fix *B N*

assume ‹$B \in \{bot\}$› **and**
‹$N \models\cap \{B\}$›
then show ‹$N - FRed_F\ N \models\cap \{B\}$›
using *Red-F-Bot consequence-relation.entails-conjunctive-def consequence-relation-axioms*
by *fastforce*
next
fix $N\ N'$ :: ‹$'f\ set$›
assume ‹$N \subseteq N'$›
then show ‹$FRed_F\ N \subseteq FRed_F\ N'$›
using *Red-F-of-subset*
by *presburger*
next
fix $N\ N'$ :: ‹$'f\ set$›
assume ‹$N \subseteq N'$›
then show ‹$FRed_I\ N \subseteq FRed_I\ N'$›
using *Red-I-of-subset*
by *presburger*
next
fix $N\ N'$
assume ‹$N' \subseteq FRed_F\ N$›
then show ‹$FRed_F\ N \subseteq FRed_F\ (N - N')$›
using *Red-F-of-Red-F-subset*
by *presburger*
next
fix $N\ N'$
assume ‹$N' \subseteq FRed_F\ N$›
then show ‹$FRed_I\ N \subseteq FRed_I\ (N - N')$›
using *Red-I-of-Red-F-subset*
by *presburger*
next
fix $\iota\ N$
assume ‹$\iota \in FInf$› **and**
‹$concl\text{-}of\ \iota \in N$›
then show ‹$\iota \in FRed_I\ N$›
using *Red-I-of-Inf-to-N*
by *blast*
next
show ‹$\{to\text{-}AF\ bot\} \neq \{\}$›
by *fast*
next
fix B :: ‹$('f, 'v)\ AF$›
assume ‹$B \in \{to\text{-}AF\ bot\}$›
then show ‹$\mathcal{G}_F\ \mathcal{J}\ B \neq \{\}$›
by (*simp add*: $\mathcal{G}_F$*-def enabled-def enabled-projection-def to-AF-def*)
next
fix B :: ‹$('f, 'v)\ AF$›
assume ‹$B \in \{to\text{-}AF\ bot\}$›
then show ‹$\mathcal{G}_F\ \mathcal{J}\ B \subseteq \{bot\}$›
using $\mathcal{G}_F$*-def enabled-projection-def*

```
    by (auto simp add: F-of-to-AF)
next
  fix ι_A
  assume ι_A-is-ainf: ‹ι_A ∈ AInf› and
         ‹(Some ∘ G_I J) ι_A ≠ None›
  have ‹G_I J ι_A ⊆ FRed_I (G_F J (concl-of ι_A))›
  proof (intro subsetI)
    fix x
    assume x-in-G_I-of-ι_A: ‹x ∈ G_I J ι_A›
    then obtain N D where ι_A-is: ‹ι_A = base-inf N D› and
                          infer-N-D-is-inf: ‹base-pre N D›
      using AInf-def ι_A-is-ainf
      by auto
    moreover have ι_A-is-enabled: ‹enabled-inf ι_A J› and
                  x-is: ‹x = ιF-of ι_A›
      using G_I-def enabled-projection-Inf-def x-in-G_I-of-ι_A
      by auto
    then have ‹prems-of ι_A = N›
      using ι_A-is
      by auto
    then have ‹fBall (fset-of-list (map A-of N)) (λ As. fset As ⊆ total-strip J)›
      using ι_A-is ι_A-is-enabled
      unfolding enabled-inf-def enabled-def
      by (simp add: fBall-fset-of-list-iff-Ball-set)
    then have ‹fset (ffUnion (A-of |`| fset-of-list N)) ⊆ total-strip J›
      by (simp add: fset-ffUnion-subset-iff-all-fsets-subset)
    then have ‹enabled (AF.Pair D (ffUnion (A-of |`| fset-of-list N))) J›
      unfolding enabled-def
      by auto
     then have ‹{AF.Pair D (ffUnion (fset-of-list (map A-of N)))} proj_J J =
{D}›
      unfolding enabled-projection-def F-of-def
      using ι_A-is-enabled ι_A-is
      by auto
   then have ‹x ∈ FRed_I (G_F J (Pair D (ffUnion (fset-of-list (map A-of N)))))›
      using x-in-G_I-of-ι_A ι_A-is-enabled x-is infer-N-D-is-inf ι_A-is
      unfolding G_I-def G_F-def ιF-of-def
      by (simp add: Red-I-of-Inf-to-N)
    then show ‹x ∈ FRed_I (G_F J (concl-of ι_A))›
      by (simp add: ι_A-is)
  qed
  then show ‹the ((Some ∘ G_I J) ι_A) ⊆ FRed_I (G_F J (concl-of ι_A))›
    by simp
next
  fix g
  show ‹po-on (⊐) UNIV›
    using tiebreaker-order-is-strict-partial-order
    by blast
next
```

fix g
show ‹*wfp-on* (⊐) *UNIV*›
using *wfp-on-tiebreaker-order*
by *blast*
qed

lemma $ARed_I$*-is-*$FRed_I$: ‹$ARed_I\ \mathcal{N} = (\bigcap\ J.$ *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}\ J\ \mathcal{N})$›
proof (*intro subset-antisym subsetI*)
fix ι
assume ‹$\iota \in ARed_I\ \mathcal{N}$›
then obtain $\mathcal{M}\ \mathcal{C}$ **where** ι*-is*: ‹$\iota =$ *base-inf* $\mathcal{M}\ \mathcal{C}$› **and**
Infer-$\mathcal{M}$*-*$\mathcal{C}$*-in-Inf*: ‹*base-pre* $\mathcal{M}\ \mathcal{C}$› **and**
ι*-in-*$FRed_I$: ‹$\forall\ \mathcal{J}.$ { *base-inf* $\mathcal{M}\ \mathcal{C}$ } $\iota proj_J\ \mathcal{J} \subseteq FRed_I\ (\mathcal{N}\ proj_J\ \mathcal{J})$›
using $ARed_I$*-def*
by *fastforce*
then have ι*-is-AInf*: ‹$\iota \in AInf$›
using *AInf-def*
by *blast*
then have ‹$\forall\ J.\ \{\iota\}\ \iota proj_J\ J \subseteq FRed_I\ (\bigcup\ (\mathcal{G}_F\ J\ `\ \mathcal{N}))$›
unfolding $\mathcal{G}_F$*-def*
using ι*-in-*$FRed_I$ ι*-is Union-of-enabled-projection-is-enabled-projection*
by *auto*
then have ‹$\forall\ J.\ \iota \in$ *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}\ J\ \mathcal{N}$›
using ι*-is-AInf*
unfolding *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}$*-def* $\mathcal{G}_I$*-def*
by *auto*
then show ‹$\iota \in (\bigcap\ J.$ *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}\ J\ \mathcal{N})$›
by *blast*
next
fix ι
assume ι*-in-*$FRed_I$*-*$\mathcal{G}$: ‹$\iota \in (\bigcap\ J.$ *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}\ J\ \mathcal{N})$›
then have ι*-is-AInf*: ‹$\iota \in AInf$› **and**
all-J-$\mathcal{G}_I$*-subset-*$FRed_I$: ‹$\forall\ J.\ \mathcal{G}_I\ J\ \iota \subseteq FRed_I\ (\bigcup\ (\mathcal{G}_F\ J\ `\ \mathcal{N}))$›
unfolding *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}$*-def*
by *auto*
then obtain $\mathcal{M}\ \mathcal{C}$ **where** ι*-is*: ‹$\iota =$ *base-inf* $\mathcal{M}\ \mathcal{C}$› **and**
Infer-$\mathcal{M}$*-*$\mathcal{C}$*-is-Inf*: ‹*base-pre* $\mathcal{M}\ \mathcal{C}$›
using *AInf-def*
by *auto*
then obtain ι_F **where** ι_F*-is*: ‹$\iota_F = \iota F$*-of* ι›
by *auto*
then have ‹$\exists\ \mathcal{M}\ \mathcal{C}.\ \iota =$ *base-inf* $\mathcal{M}\ \mathcal{C} \wedge$ *base-pre* $\mathcal{M}\ \mathcal{C} \wedge$
$(\forall\ \mathcal{J}.$ { *base-inf* $\mathcal{M}\ \mathcal{C}$ } $\iota proj_J\ \mathcal{J} \subseteq FRed_I\ (\mathcal{N}\ proj_J\ \mathcal{J}))$›
using ι*-is Infer-*$\mathcal{M}$*-*$\mathcal{C}$*-is-Inf all-J-*$\mathcal{G}_I$*-subset-*$FRed_I$
unfolding $\mathcal{G}_I$*-def* $\mathcal{G}_F$*-def*
using *Union-of-enabled-projection-is-enabled-projection*
by *fastforce*

then show ‹$\iota \in ARed_I\ \mathcal{N}$›
unfolding $ARed_I$-*def*
by *auto*
qed

lemma $ARed_F$-*is-*$FRed_F$: ‹$ARed_F\ \mathcal{N} = (\bigcap\ J.$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}\ J\ \mathcal{N})$›
proof (*intro subset-antisym subsetI*)
fix $\mathcal{C}$
assume $\mathcal{C}$-*in-*$ARed_F$: ‹$\mathcal{C} \in ARed_F\ \mathcal{N}$›
then obtain $C\ A$ **where** $\mathcal{C}$-*is*: ‹$\mathcal{C} = AF.Pair\ C\ A$›
unfolding $ARed_F$-*def* $SRed_F$-*def*
by *blast*
consider (*a*) ‹$\forall\ \mathcal{J}.$ *fset* $A \subseteq$ *total-strip* $\mathcal{J} \longrightarrow C \in FRed_F\ (\mathcal{N}\ proj_J\ \mathcal{J})$› |
(*b*) ‹$\exists\ \mathcal{C} \in \mathcal{N}.$ *F-of* $\mathcal{C} = C \wedge$ *A-of* $\mathcal{C}\ |\subset|\ A$›
using $\mathcal{C}$-*in-*$ARed_F$ $\mathcal{C}$-*is*
unfolding $ARed_F$-*def* $SRed_F$-*def*
by *blast*
then show ‹$\mathcal{C} \in (\bigcap\ J.$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}\ J\ \mathcal{N})$›
unfolding *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}$-*def*
proof (*cases*)
case *a*
then have ‹$\forall\ J.\ \forall\ D \in \mathcal{G}_F\ J\ \mathcal{C}.\ D \in FRed_F\ (\bigcup\ (\mathcal{G}_F\ J\ `\ \mathcal{N}))$›
unfolding Red_F-*strict-def* $\mathcal{G}_F$-*def*
using *Union-of-enabled-projection-is-enabled-projection* $\mathcal{C}$-*is enabled-projection-def*
$\mathcal{C}$-*is complete enabled-projection-def*
using *enabled-def*
by *force*
then have ‹$\mathcal{C} \in (\bigcap\ J.\ \{\ C.\ \forall\ D \in \mathcal{G}_F\ J\ C.\ D \in FRed_F\ (\bigcup\ (\mathcal{G}_F\ J\ `\ \mathcal{N}))\ \})$›
by *blast*
then show ‹$\mathcal{C} \in (\bigcap\ J.\ \{\ C.\ \forall\ D \in \mathcal{G}_F\ J\ C.\ D \in FRed_F\ (\bigcup\ (\mathcal{G}_F\ J\ `\ \mathcal{N})) \vee$
$(\exists\ E \in \mathcal{N}.\ E \sqsupset C \wedge D \in \mathcal{G}_F\ J\ E)\ \})$›
by *blast*
next
case *b*
then have ‹$\forall\ J.\ \forall\ D \in \mathcal{G}_F\ J\ \mathcal{C}.\ \exists\ E \in \mathcal{N}.\ E \sqsupset \mathcal{C} \wedge D \in \mathcal{G}_F\ J\ E$›
unfolding $\mathcal{G}_F$-*def tiebreaker-order-def enabled-projection-def*
using *subformula-of-enabled-formula-is-enabled*

by (*smt* (*verit*, *ccfv-SIG*) *AF.sel*(*1*) *AF.sel*(*2*) $\mathcal{C}$-*is case-prodI mem-Collect-eq*
singletonD singletonI)
then have ‹$\mathcal{C} \in (\bigcap\ J.\ \{\ C.\ \forall\ D \in \mathcal{G}_F\ J\ C.\ \exists\ E \in \mathcal{N}.\ E \sqsupset C \wedge D \in \mathcal{G}_F\ J\ E$
$\})$›
by *blast*
then show ‹$\mathcal{C} \in (\bigcap\ J.\ \{\ C.\ \forall\ D \in \mathcal{G}_F\ J\ C.\ D \in FRed_F\ (\bigcup\ (\mathcal{G}_F\ J\ `\ \mathcal{N})) \vee$
$(\exists\ E \in \mathcal{N}.\ E \sqsupset C \wedge D \in \mathcal{G}_F\ J\ E)\ \})$›
by *blast*
qed

next
fix $\mathcal{C}$
assume $\mathcal{C}$*-in-FRed*$_F$-$\mathcal{G}$: ‹$\mathcal{C} \in (\bigcap\ J.$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}$ $J\ \mathcal{N})$›
then have $\mathcal{C}$*-in-FRed*$_F$-$\mathcal{G}$*-unfolded*:
‹$\forall\ J.\ \forall\ D \in \mathcal{G}_F\ J\ \mathcal{C}.\ D \in FRed_F\ (\bigcup\ (\mathcal{G}_F\ J\ `\ \mathcal{N})) \vee (\exists\ E \in \mathcal{N}.\ E \sqsupset \mathcal{C} \wedge D \in \mathcal{G}_F\ J\ E)$›
unfolding *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}$*-def*
by *blast*
then have $\mathcal{C}$*-in-FRed*$_F$-$\mathcal{G}$*-if-enabled*:
‹$\forall\ J.$ *enabled* $\mathcal{C}\ J \longrightarrow$ *F-of* $\mathcal{C} \in FRed_F\ (\bigcup\ (\mathcal{G}_F\ J\ `\ \mathcal{N})) \vee (\exists\ E \in \mathcal{N}.\ E \sqsupset \mathcal{C} \wedge$ *F-of* $\mathcal{C} \in \mathcal{G}_F\ J\ E)$›
unfolding $\mathcal{G}_F$*-def enabled-projection-def*
by *auto*
obtain $C\ A$ **where** $\mathcal{C}$*-is*: ‹$\mathcal{C} = AF.Pair\ C\ A$›
by (*meson AF.exhaust-sel*)
then have
‹$\forall\ J.$ *fset* $A \subseteq$ *total-strip* $J \longrightarrow$
$C \in FRed_F\ (\bigcup\ (\mathcal{G}_F\ J\ `\ \mathcal{N})) \vee (\exists\ E \in \mathcal{N}.\ E \sqsupset \mathcal{C} \wedge C \in \mathcal{G}_F\ J\ E)$›
using $\mathcal{C}$*-in-FRed*$_F$-$\mathcal{G}$*-if-enabled*
unfolding *enabled-def*
by *simp*
then show ‹$\mathcal{C} \in ARed_F\ \mathcal{N}$›
using $\mathcal{C}$*-is* $\mathcal{C}$*-in-FRed*$_F$-$\mathcal{G}$*-if-enabled*
unfolding *ARed*$_F$*-def SRed*$_F$*-def* $\mathcal{G}_F$*-def enabled-def tiebreaker-order-def*
using *Union-of-enabled-projection-is-enabled-projection*
by *auto*
qed

lemma *entails-is-entails-*$\mathcal{G}$: ‹$\mathcal{M} \models_{AF} \{\mathcal{C}\} \longleftrightarrow (\forall\ \mathcal{J}.$ *lift-from-FRed-to-ARed.entails-*$\mathcal{G}$ $\mathcal{J}\ \mathcal{M}\ \{\mathcal{C}\})$›
proof (*intro iffI allI*)
fix $\mathcal{J}$
assume ‹$\mathcal{M} \models_{AF} \{\mathcal{C}\}$›
then show ‹*lift-from-FRed-to-ARed.entails-*$\mathcal{G}$ $\mathcal{J}\ \mathcal{M}\ \{\mathcal{C}\}$›
unfolding $\mathcal{G}_F$*-def AF-entails-def enabled-projection-def enabled-set-def entails-conjunctive-def*
by (*simp add*: *Union-of-singleton-is-setcompr*)
next
assume *entails-*$\mathcal{G}$-$\mathcal{M}$-$\mathcal{C}$: ‹$\forall\ \mathcal{J}.$ *lift-from-FRed-to-ARed.entails-*$\mathcal{G}$ $\mathcal{J}\ \mathcal{M}\ \{\mathcal{C}\}$›
show ‹$\mathcal{M} \models_{AF} \{\mathcal{C}\}$›
unfolding $\mathcal{G}_F$*-def AF-entails-def enabled-set-def*
proof (*intro allI impI*)
fix J
assume ‹$\forall\ \mathcal{C} \in \{\mathcal{C}\}.$ *enabled* $\mathcal{C}\ J$›
then show ‹$\mathcal{M}$ *proj*$_J$ $J \models$ *F-of* $`\ \{\mathcal{C}\}$›
using *entails-*$\mathcal{G}$-$\mathcal{M}$-$\mathcal{C}$
unfolding $\mathcal{G}_F$*-def enabled-projection-def entails-conjunctive-def*
by (*simp add*: *Union-of-singleton-is-setcompr*)

```
    qed
qed



lemma SRed_I-in-SInf: ‹SRed_I N ⊆ SInf›
  using SRed_I-def SInf.simps
  by force

lemma SRed_F-entails-bot: ‹N ⊨_AF {to-AF bot} ⟹ N − SRed_F N ⊨_AF {to-AF
bot}›
proof −
  fix N

  have And-to-Union:
   ‹⋀ J. N − lift-from-FRed-to-ARed.Red-F-G J N ⊆ (⋃ J. N − lift-from-FRed-to-ARed.Red-F-G
J N)›
    by blast

  assume N-entails-bot: ‹N ⊨_AF {to-AF bot}›
  have ‹lift-from-FRed-to-ARed.entails-G J N {to-AF bot} ⟹
        lift-from-FRed-to-ARed.entails-G J (N − lift-from-FRed-to-ARed.Red-F-G
J N) {to-AF bot}›
    for J
    using lift-from-FRed-to-ARed.Red-F-Bot-F
    by blast
  then have ‹N ⊨_AF {to-AF bot} ⟹ N − ARed_F N ⊨_AF {to-AF bot}›
  proof −
    assume ‹N ⊨_AF {to-AF bot}› and
          ‹⋀ J. lift-from-FRed-to-ARed.entails-G J N {to-AF bot} ⟹
          lift-from-FRed-to-ARed.entails-G J (N − lift-from-FRed-to-ARed.Red-F-G
J N) {to-AF bot}›
    then have FRed_F-G-entails-G-bot:
      ‹lift-from-FRed-to-ARed.entails-G J (N − lift-from-FRed-to-ARed.Red-F-G J
N) {to-AF bot}›
      for J
      using entails-is-entails-G
      by blast
    then have
    ‹lift-from-FRed-to-ARed.entails-G J (⋃ J. N − lift-from-FRed-to-ARed.Red-F-G
J N) {to-AF bot}›
      for J
      using And-to-Union
    by (meson lift-from-FRed-to-ARed.entails-trans lift-from-FRed-to-ARed.subset-entailed)
    then show ‹N − ARed_F N ⊨_AF {to-AF bot}›
      using ARed_F-is-FRed_F entails-is-entails-G
      by fastforce
  qed
  then show ‹N − SRed_F N ⊨_AF {to-AF bot}›
```

using $ARed_F$*-def N-entails-bot*
by *force*
qed

lemma $SRed_F$*-of-subset-F*: ‹$N \subseteq N' \Longrightarrow SRed_F\ N \subseteq SRed_F\ N'$›
proof −
fix $N\ N'$ :: ‹$('f, 'v)\ AF\ set$›
assume ‹$N \subseteq N'$›
then show ‹$SRed_F\ N \subseteq SRed_F\ N'$›
unfolding $SRed_F$*-def enabled-projection-def*

by *auto*
(*smt* (*verit*, *best*) *Collect-mono Red-F-of-subset subset-iff*)
qed

lemma $SRed_I$*-of-subset-F*: ‹$N \subseteq N' \Longrightarrow SRed_I\ N \subseteq SRed_I\ N'$›
proof −
fix $N\ N'$ :: ‹$('f, 'v)\ AF\ set$›
assume ‹$N \subseteq N'$›
then show ‹$SRed_I\ N \subseteq SRed_I\ N'$›
unfolding $SRed_I$*-def enabled-projection-Inf-def enabled-projection-def enabled-inf-def* *ιF-of-def*

by (*auto*, (*smt* (*verit*, *best*) *Red-I-of-subset mem-Collect-eq subset-iff*)+)
qed

lemma $SRed_F$*-of-*$SRed_F$*-subset-F*: ‹$N' \subseteq SRed_F\ N \Longrightarrow SRed_F\ N \subseteq SRed_F\ (N - N')$›
proof −
fix $N\ N'$
assume N'*-subset-*$SRed_F$*-N*: ‹$N' \subseteq SRed_F\ N$›
have ‹$N' \subseteq ARed_F\ N \Longrightarrow ARed_F\ N \subseteq ARed_F\ (N - N')$›
using *lift-from-FRed-to-ARed.Red-F-of-Red-F-subset-F*
proof −
assume N'*-subset-*$ARed_F$*-N*: ‹$N' \subseteq ARed_F\ N$› **and**
‹$(\bigwedge N'\ \mathcal{J}\ N.\ N' \subseteq$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}\ \mathcal{J}\ N \Longrightarrow$
lift-from-FRed-to-ARed.Red-F-$\mathcal{G}\ \mathcal{J}\ N \subseteq$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}\ \mathcal{J}\ (N - N'))$›
then have ‹$\bigwedge N'\ N.\ N' \subseteq (\bigcap \mathcal{J}.$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}\ \mathcal{J}\ N) \Longrightarrow$
$(\bigcap \mathcal{J}.$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}\ \mathcal{J}\ N) \subseteq$
$(\bigcap \mathcal{J}.$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}\ \mathcal{J}\ (N - N'))$›
by (*meson INF-mono' UNIV-I le-INF-iff*)
then show ‹$ARed_F\ N \subseteq ARed_F\ (N - N')$›
using $ARed_F$*-is-*$FRed_F$ N'*-subset-*$ARed_F$*-N*
by *presburger*
qed
then show ‹$SRed_F\ N \subseteq SRed_F\ (N - N')$›
by (*simp add*: $ARed_F$*-def* N'*-subset-*$SRed_F$*-N*)
qed

lemma $SRed_I$*-of-*$SRed_F$*-subset-F*: ‹$N' \subseteq SRed_F\ N \Longrightarrow SRed_I\ N \subseteq SRed_I\ (N - N')$›
proof −
fix $N\ N'$
assume N'*-subset-*$SRed_F$*-N*: ‹$N' \subseteq SRed_F\ N$›
have *works-for-*$ARed_I$: ‹$N' \subseteq ARed_F\ N \Longrightarrow ARed_I\ N \subseteq ARed_I\ (N - N')$›
using *lift-from-FRed-to-ARed.Red-I-of-Red-F-subset-F*
proof −
assume N'*-subset-*$ARed_F$*-N*: ‹$N' \subseteq ARed_F\ N$› **and**
‹$(\bigwedge N'\ \mathcal{J}\ N.\ N' \subseteq$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}\ \mathcal{J}\ N \Longrightarrow$
lift-from-FRed-to-ARed.Red-I-$\mathcal{G}\ \mathcal{J}\ N \subseteq$ *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}\ \mathcal{J}\ (N - N'))$›
then have ‹$\bigwedge N'\ N.\ N' \subseteq (\bigcap \mathcal{J}.$ *lift-from-FRed-to-ARed.Red-F-*$\mathcal{G}\ \mathcal{J}\ N) \Longrightarrow$
$(\bigcap \mathcal{J}.$ *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}\ \mathcal{J}\ N) \subseteq$
$(\bigcap \mathcal{J}.$ *lift-from-FRed-to-ARed.Red-I-*$\mathcal{G}\ \mathcal{J}\ (N - N'))$›
by (*metis* (*no-types*, *lifting*) *INF-mono'* *UNIV-I* *le-INF-iff*)
then show ‹$ARed_I\ N \subseteq ARed_I\ (N - N')$›
using $ARed_I$*-is-*$FRed_I$ $ARed_F$*-is-*$FRed_F$ N'*-subset-*$ARed_F$*-N*
by *presburger*
qed
moreover have ‹*unsat-pre* $\mathcal{N} \Longrightarrow$ *unsat-inf* $\mathcal{N} \in SRed_I\ (N - N')$›
if ι*-is-redundant*: ‹*unsat-inf* $\mathcal{N} \in SRed_I\ N$›
for $\mathcal{N}$
using *bot-not-in-sredF-*$\mathcal{N}$ N'*-subset-*$SRed_F$*-N* ι*-is-redundant*
unfolding $SRed_I$*-def* $SRed_F$*-def*

by (*smt* (*verit*, *del-insts*) $ARed_F$*-def* $ARed_I$*-def* *Diff-iff* N'*-subset-*$SRed_F$*-N* *Un-iff*
bot-not-in-sredF-$\mathcal{N}$ *works-for-*$ARed_I$ *mem-Collect-eq* *subsetD*)
ultimately show ‹$SRed_I\ N \subseteq SRed_I\ (N - N')$›
using N'*-subset-*$SRed_F$*-N* *bot-not-in-sredF-*$\mathcal{N}$
unfolding $SRed_F$*-def* $ARed_F$*-def* $SRed_I$*-def* $ARed_I$*-def*

by (*smt* (*verit*, *del-insts*) *Collect-cong* *Diff-iff* N'*-subset-*$SRed_F$*-N* *Un-iff*
bot-not-in-sredF-$\mathcal{N}$ *subset-iff*)
qed

lemma $SRed_I$*-of-SInf-to-N-F*: ‹$\iota_S \in SInf \Longrightarrow$ *concl-of* $\iota_S \in N \Longrightarrow \iota_S \in SRed_I\ N$›
proof −
fix $\iota_S\ N$
assume ‹$\iota_S \in SInf$› **and**
concl-ι_S*-in-N*: ‹*concl-of* $\iota_S \in N$›
then show ‹$\iota_S \in SRed_I\ N$›
unfolding $SRed_I$*-def*
proof (*cases* ι_S *rule*: *SInf.cases*)
case (*base* $\mathcal{N}$ *D*)
obtain $\mathcal{M}\ \mathcal{C}$ **where** ι_S*-is*: ‹ι_S = *base-inf* $\mathcal{M}\ \mathcal{C}$› **and**

```
                Infer-M-C-is-Inf: ‹base-pre M C›
    using base
    by blast
  have ‹∀ J. { base-inf M C } ιproj_J J ⊆ FRed_I (N proj_J J)›
      unfolding enabled-projection-Inf-def enabled-projection-def ιF-of-def en-
abled-inf-def
  proof (intro allI subsetI)
    fix J
    have ‹∀C∈set M. enabled C J ⟹ Infer (map F-of M) C ∈ FRed_I {F-of C
|C. C ∈ N ∧ enabled C J}›
    proof −
      assume all-enabled-in-M: ‹∀ C ∈ set M. enabled C J›
      then have A-of-M-to-C-in-N: ‹AF.Pair C (ffUnion (fset-of-list (map A-of
M))) ∈ N›
        using ι_S-is concl-ι_S-in-N
        by auto
      moreover have ‹fBall (fset-of-list M) (λ x. fset (A-of x) ⊆ total-strip J)›
        using all-enabled-in-M
        unfolding enabled-def
        by (simp add: fset-of-list-elem)
      then have ‹fBall (A-of |‘| fset-of-list M) (λ x. fset x ⊆ total-strip J)›
        by auto
      then have ‹enabled (AF.Pair C (ffUnion (A-of |‘| fset-of-list M))) J›
        using A-of-M-to-C-in-N
        unfolding enabled-def
        using fset-ffUnion-subset-iff-all-fsets-subset
        by (metis AF.sel(2))
      ultimately show ‹Infer (map F-of M) C ∈ FRed_I {F-of C | C. C ∈ N ∧
enabled C J}›
      by (metis (mono-tags, lifting) AF.sel(1) Infer-M-C-is-Inf Red-I-of-Inf-to-N
          fset-of-list-map inference.sel(2) mem-Collect-eq)
    qed
    then show ‹x ∈ {Infer (map F-of (prems-of ι)) (F-of (concl-of ι)) |ι.
          ι ∈ {base-inf M C} ∧ (∀C∈set (prems-of ι). enabled C J)} ⟹
        x ∈ FRed_I {F-of C |C. C ∈ N ∧ enabled C J}› for x
      by simp
  qed
  then have ‹ι_S ∈ { base-inf M C | M C. base-pre M C ∧
                (∀ J. { base-inf M C } ιproj_J J ⊆ FRed_I (N proj_J J))}›
    using ι_S-is Infer-M-C-is-Inf
    by auto
  then show ‹ι_S ∈ { base-inf M C | M C. base-pre M C ∧
                (∀ J. { base-inf M C } ιproj_J J ⊆ FRed_I (N proj_J J)) } ∪
              { unsat-inf M | M. unsat-pre M ∧ to-AF bot ∈ N }›
    by fast
next
  case (unsat N)
  then have ‹ι_S ∈ { unsat-inf M | M. unsat-pre M ∧ to-AF bot ∈ N}›
    using concl-ι_S-in-N
```

by *fastforce*
then show ‹$\iota_S \in \{$ *base-inf* $\mathcal{M}$ $\mathcal{C}$ | $\mathcal{M}$ $\mathcal{C}$. *base-pre* $\mathcal{M}$ $\mathcal{C}$ $\wedge$
($\forall$ $\mathcal{J}$. $\{$ *base-inf* $\mathcal{M}$ $\mathcal{C}$ $\}$ $\iota proj_J$ $\mathcal{J} \subseteq$ *FRed*$_I$ (*N proj*$_J$ $\mathcal{J}$)) $\} \cup$
$\{$ *unsat-inf* $\mathcal{M}$ | $\mathcal{M}$. *unsat-pre* $\mathcal{M}$ $\wedge$ *to-AF bot* $\in$ *N* $\}$›
by *fast*
qed
qed

end

8.3 Standard completeness

context *annotated-calculus*
begin

lemmas *SRed-rules* = *SRed*$_F$*-entails-bot SRed*$_F$*-of-subset-F SRed*$_I$*-of-subset-F SRed*$_F$*-of-SRed*$_F$*-subset-F*
SRed$_I$*-of-SRed*$_F$*-subset-F SRed*$_I$*-of-SInf-to-N-F SRed*$_I$*-in-SInf*

sublocale *S-calculus*: *calculus* ‹*to-AF bot*› *SInf AF-entails SRed*$_I$ *SRed*$_F$
by (*standard*; *simp add*: *SRed-rules*)

lemma *S-saturated-to-F-saturated*: ‹*S-calculus.saturated* $\mathcal{N} \Longrightarrow$ *saturated* ($\mathcal{N}$ *proj*$_J$ $\mathcal{J}$)›
proof −
assume $\mathcal{N}$*-is-S-saturated*: ‹*S-calculus.saturated* $\mathcal{N}$›
then show ‹*saturated* ($\mathcal{N}$ *proj*$_J$ $\mathcal{J}$)›
unfolding *saturated-def S-calculus.saturated-def*
proof (*intro subsetI*)
fix ι_F
assume ‹$\iota_F \in$ *Inf-from* ($\mathcal{N}$ *proj*$_J$ $\mathcal{J}$)›
then have ι_F*-is-Inf*: ‹$\iota_F \in$ *FInf*› **and**
prems-of-ι_F*-in-*$\mathcal{N}$*-proj-*$\mathcal{J}$: ‹*set* (*prems-of* ι_F) $\subseteq$ $\mathcal{N}$ *proj*$_J$ $\mathcal{J}$›
unfolding *Inf-from-def*
by *auto*
moreover have ‹$\forall$ $C \in$ *set* (*prems-of* ι_F). $\exists$ $\mathcal{C} \in \mathcal{N}$. *F-of* $\mathcal{C}$ = C $\wedge$ *enabled* $\mathcal{C}$ $\mathcal{J}$›
using *prems-of-*ι_F*-in-*$\mathcal{N}$*-proj-*$\mathcal{J}$
unfolding *enabled-projection-def*
by *blast*
then have ‹*list-all* (λ C. $\exists$ $\mathcal{C} \in \mathcal{N}$. *F-of* $\mathcal{C}$ = C $\wedge$ *enabled* $\mathcal{C}$ $\mathcal{J}$) (*prems-of* ι_F)›
using *Ball-set*
by *blast*
then have ‹$\exists$ $\mathcal{C}s$. *length* $\mathcal{C}s$ = *length* (*prems-of* ι_F) $\wedge$
list-all2 (λ C $\mathcal{C}$. $\mathcal{C} \in \mathcal{N}$ $\wedge$ *F-of* $\mathcal{C}$ = C $\wedge$ *enabled* $\mathcal{C}$ $\mathcal{J}$) (*prems-of* ι_F) $\mathcal{C}s$›
using *list-all-ex-to-ex-list-all2*

by (*smt* (*verit*, *best*) *Ball-set*)
then have ‹∃ *As*. *length As* = *length* (*prems-of* ι_F) ∧
list-all2 (λ *C A*. *AF.Pair C A* ∈ $\mathcal{N}$ ∧ *enabled* (*AF.Pair C A*) $\mathcal{J}$)
(*prems-of* ι_F) *As*›
by (*smt* (*verit*, *del-insts*) *AF.exhaust AF.sel*(*1*) *list.pred-mono-strong*
list-all-ex-to-ex-list-all2)
then have ‹∃ *As*. *length As* = *length* (*prems-of* ι_F) ∧
list-all (λ *C*. *C* ∈ $\mathcal{N}$ ∧ *enabled C* $\mathcal{J}$) (*map2 AF.Pair* (*prems-of*
ι_F) *As*)›
using *list-all2-to-map*[**where** *f* = ‹λ *C A*. *AF.Pair C A*›]
by (*smt* (*verit*) *list-all2-mono*)
then obtain *As* :: ‹*'v sign fset list*›
where ‹∀ *C* ∈ *set* (*map2 AF.Pair* (*prems-of* ι_F) *As*). *C* ∈ $\mathcal{N}$ ∧
enabled C $\mathcal{J}$› **and**
length-As-eq-length-prems: ‹*length As* = *length* (*prems-of* ι_F)›
by (*metis* (*no-types*, *lifting*) *Ball-set-list-all*)
then have *set-prems-As-subset-*$\mathcal{N}$: ‹*set* (*map2 AF.Pair* (*prems-of* ι_F) *As*) ⊆
$\mathcal{N}$› **and**
all-enabled: ‹∀ *C* ∈ *set* (*map2 AF.Pair* (*prems-of* ι_F) *As*). *enabled C*
$\mathcal{J}$›
by *auto*

let *?prems* = ‹*map2 AF.Pair* (*prems-of* ι_F) *As*›

have ‹*set ?prems* ⊆ $\mathcal{N}$›
using *set-prems-As-subset-*$\mathcal{N}$ **.**
moreover have ‹*length ?prems* = *length* (*prems-of* ι_F)›
using *length-As-eq-length-prems*
by *simp*
then have *F-of-dummy-prems-is-prems-of-*ι_F: ‹*map F-of ?prems* = *prems-of*
ι_F›
by (*simp add*: *length-As-eq-length-prems*)
moreover have ‹∀ *C* ∈ *set* (*map A-of* (*map2 AF.Pair* (*prems-of* ι_F) *As*)).
fset C ⊆ *total-strip* $\mathcal{J}$›
using
all-enabled ball-set-f-to-ball-set-map[**where** *P* = ‹λ *x*. *fset x* ⊆ *total-strip* $\mathcal{J}$›
and *f* = *A-of*]
unfolding *enabled-def*
by *blast*
then have ‹∀ *C* ∈ *set As*. *fset C* ⊆ *total-strip* $\mathcal{J}$›
using *map-A-of-map2-Pair length-As-eq-length-prems*
by *metis*
then have ‹*fset* (*ffUnion* (*fset-of-list As*)) ⊆ *total-strip* $\mathcal{J}$›
using *all-enabled*
unfolding *enabled-def*[*of* - $\mathcal{J}$]
by (*simp add*: *fBall-fset-of-list-iff-Ball-set fset-ffUnion-subset-iff-all-fsets-subset*)
then have *base-inf-enabled*: ‹*enabled-inf* (*base-inf ?prems* (*concl-of* ι_F)) $\mathcal{J}$›
using *all-enabled enabled-inf-def*
by *auto*

moreover have *pre-holds*: ‹*base-pre ?prems* (*concl-of* ι_F)›
using ι_F*-is-Inf F-of-dummy-prems-is-prems-of-*ι_F
by *force*
moreover have ι*F-of-base-inf-is-*ι_F: ‹ι*F-of* (*base-inf ?prems* (*concl-of* ι_F)) = ι_F›
using *F-of-dummy-prems-is-prems-of-*ι_F ι*F-of-def*
by *force*
ultimately have ι_F*-in-Inf-*$\mathcal{N}$*-proj-*$\mathcal{J}$: ‹ι_F ∈ (*S-calculus.Inf-from* $\mathcal{N}$) $\iota proj_J$ $\mathcal{J}$›
using *SInf.base*[*OF pre-holds*]
unfolding *enabled-projection-Inf-def S-calculus.Inf-from-def*
by (*metis* (*mono-tags*, *lifting*) *inference.sel*(*1*) *mem-Collect-eq*)
then have ‹∃ $\mathcal{M}$ *D*. *base-inf* $\mathcal{M}$ *D* ∈ *S-calculus.Inf-from* $\mathcal{N}$ ∧
ι*F-of* (*base-inf* $\mathcal{M}$ *D*) = ι_F ∧ *enabled-inf* (*base-inf* $\mathcal{M}$ *D*) $\mathcal{J}$›
using ι*F-of-base-inf-is-*ι_F
unfolding *enabled-projection-Inf-def*
by (*metis* (*mono-tags*, *lifting*) *CollectI S-calculus.Inf-from-def SInf.base base-inf-enabled inference.sel*(*1*) *pre-holds set-prems-As-subset-*$\mathcal{N}$)
then obtain $\mathcal{M}$ *D* **where** *base-inf-in-Inf-*$\mathcal{N}$: ‹*base-inf* $\mathcal{M}$ *D* ∈ *S-calculus.Inf-from* $\mathcal{N}$› **and**
ι*F-of-base-inf-is-*ι_F: ‹ι*F-of* (*base-inf* $\mathcal{M}$ *D*) = ι_F› **and**
base-inf-enabled: ‹*enabled-inf* (*base-inf* $\mathcal{M}$ *D*) $\mathcal{J}$›
by *blast*
then have ‹*base-inf* $\mathcal{M}$ *D* ∈ $SRed_I$ $\mathcal{N}$›
using $\mathcal{N}$*-is-S-saturated*
unfolding *S-calculus.saturated-def*
by *blast*
moreover have ‹*base-pre* $\mathcal{M}$ *D*›
using ι*F-of-base-inf-is-*ι_F ι_F*-is-Inf*
by (*simp add*: ι*F-of-def*)
ultimately show ‹ι_F ∈ $FRed_I$ ($\mathcal{N}$ $proj_J$ $\mathcal{J}$)›
using ι_F*-in-Inf-*$\mathcal{N}$*-proj-*$\mathcal{J}$ ι*F-of-base-inf-is-*ι_F *base-inf-enabled*
unfolding $SRed_I$*-def enabled-projection-Inf-def* ι*F-of-def enabled-def enabled-projection-def*
by *auto*
(*metis* (*mono-tags*, *lifting*) *AF.sel*(*2*) *F-of-to-AF Red-I-of-Inf-to-N bot-fset.rep-eq empty-subsetI inference.sel*(*2*) *mem-Collect-eq to-AF-def*)
qed
qed

notation *AF-cons-rel.entails-conjunctive* (**infix** ‹$\models\cap_{AF}$› *50*)

theorem *S-calculus-statically-complete*:
assumes *F-statically-complete*: ‹*statically-complete-calculus bot FInf* ($\models$) $FRed_I$ $FRed_F$›
shows ‹*statically-complete-calculus* (*to-AF bot*) *SInf* ($\models_{AF}$) $SRed_I$ $SRed_F$›
using *F-statically-complete*
unfolding *statically-complete-calculus-def statically-complete-calculus-axioms-def*

proof (*intro conjI allI impI*; *elim conjE*)
show ‹*calculus* (*to-AF bot*) *SInf* ($\models_{AF}$) *SRed*$_I$ *SRed*$_F$›
using *S-calculus.calculus-axioms*
by *force*
next
fix *N*
assume ‹*calculus bot FInf* ($\models$) *FRed*$_I$ *FRed*$_F$› **and**
if-F-saturated-and-N-entails-bot-then-bot-in-N:
‹$\forall$ *N*. *saturated N* $\longrightarrow$ *N* $\models$ {*bot*} $\longrightarrow$ *bot* $\in$ *N*› **and**
N-is-S-saturated: ‹*S-calculus.saturated N*› **and**
N-entails-bot: ‹*N* $\models_{AF}$ {*to-AF bot*}›
then have *N-proj-*$\mathcal{J}$*-entails-bot*: ‹$\forall$ $\mathcal{J}$. *N proj*$_J$ $\mathcal{J}$ $\models$ {*bot*}›
unfolding *AF-entails-def*
using *F-of-to-AF*[*of bot*]
by (*smt* (*verit*) *enabled-to-AF-set image-empty image-insert*)
then have *N-proj-*$\mathcal{J}$*-F-saturated*: ‹$\forall$ $\mathcal{J}$. *saturated* (*N proj*$_J$ $\mathcal{J}$)›
using *N-is-S-saturated*
using *S-saturated-to-F-saturated*
by *blast*
then have ‹$\forall$ $\mathcal{J}$. *bot* $\in$ *N proj*$_J$ $\mathcal{J}$›
using *N-proj-*$\mathcal{J}$*-entails-bot if-F-saturated-and-N-entails-bot-then-bot-in-N*
by *presburger*
then have *prop-proj-N-is-prop-unsat*: ‹*propositionally-unsatisfiable* (*proj*$_\perp$ *N*)›
unfolding *enabled-projection-def propositional-model-def propositional-projection-def*
propositionally-unsatisfiable-def
by *fast*
then have ‹*proj*$_\perp$ *N* $\neq$ {}›
unfolding *propositionally-unsatisfiable-def propositional-model-def*
using *enabled-projection-def prop-proj-in*
by *auto*
then have ‹$\exists$ $\mathcal{M}$. *set* $\mathcal{M}$ $\subseteq$ *proj*$_\perp$ *N* $\wedge$ *finite* (*set* $\mathcal{M}$) $\wedge$ *propositionally-unsatisfiable* (*set* $\mathcal{M}$)›
by (*metis finite-list prop-proj-N-is-prop-unsat prop-unsat-compactness*)
then obtain $\mathcal{M}$ **where** $\mathcal{M}$*-subset-prop-proj-N*: ‹*set* $\mathcal{M}$ $\subseteq$ *proj*$_\perp$ *N*› **and**
$\mathcal{M}$*-subset-N*: ‹*set* $\mathcal{M}$ $\subseteq$ *N*› **and**
‹*finite* (*set* $\mathcal{M}$)› **and**
$\mathcal{M}$*-prop-unsat*: ‹*propositionally-unsatisfiable* (*set* $\mathcal{M}$)› **and**
$\mathcal{M}$*-not-empty*: ‹$\mathcal{M}$ $\neq$ []›
by (*smt* (*verit*, *del-insts*) *AF-cons-rel.entails-bot-to-entails-empty*
AF-cons-rel.entails-empty-reflexive-dangerous compactness-AF-proj equiv-prop-entails
finite-list image-empty prop-proj-N-is-prop-unsat prop-proj-in propositional-model2-def
propositionally-unsatisfiable-def set-empty2 subset-empty subset-trans to-AF-proj-J)
then have ‹*unsat-inf* $\mathcal{M}$ $\in$ *S-calculus.Inf-from N*› **and**
Infer-$\mathcal{M}$*-bot-in-SInf*: ‹*unsat-inf* $\mathcal{M}$ $\in$ *SInf*›
using *SInf.unsat S-calculus.Inf-from-def propositional-projection-def*
by *fastforce+*
then have ‹*unsat-inf* $\mathcal{M}$ $\in$ *SRed*$_I$ *N*›
using *N-is-S-saturated S-calculus.saturated-def*
by *blast*

```
  then show ‹to-AF bot ∈ N›
    unfolding SRed_I-def
  proof (elim UnE)
    assume ‹unsat-inf M ∈ { base-inf M C | M C. base-pre M C ∧
      (∀ J. { base-inf M C } ιproj_J J ⊆ FRed_I (N proj_J J)) }›
    then have ‹unsat-inf M = base-inf M bot›
      by (smt (verit, best) AF.exhaust-sel AF.sel(2) F-of-to-AF inference.inject
mem-Collect-eq)
    then have ‹to-AF bot = AF.Pair bot (ffUnion (A-of |‘| fset-of-list M))›
      by simp
    then have ‹ffUnion (A-of |‘| fset-of-list M) = {||}›
      by (metis AF.sel(2) A-of-to-AF)
    then consider (M-empty) ‹A-of |‘| fset-of-list M = {||}› |
                  (no-assertions-in-M) ‹fBall (A-of |‘| fset-of-list M) (λ x. x = {||})›
      using Union-empty-if-set-empty-or-all-empty
      by auto
    then show ?thesis
    proof (cases)
      case M-empty
      then have ‹fset-of-list M = {||}›
        by blast
      then have ‹M = []›
        by (metis bot-fset.rep-eq fset-of-list.rep-eq set-empty2)
      then show ?thesis
        using M-not-empty
        by contradiction
    next
      case no-assertions-in-M
      then have ‹fBall (fset-of-list M) (λ x. A-of x = {||})›
        using fBall-fimage-is-fBall
        by simp
      then have ‹∀ x ∈ set M. A-of x = {||}›
        using fBall-fset-of-list-iff-Ball-set
        by fast
      then have ‹to-AF bot ∈ set M›
        using M-subset-prop-proj-N M-not-empty
        unfolding propositional-projection-def to-AF-def
       by (metis (mono-tags, lifting) AF.exhaust-sel CollectD ex-in-conv set-empty
subset-code(1))
      then show ?thesis
        using M-subset-N
        by blast
    qed
  next
    assume ‹unsat-inf M ∈ { unsat-inf M | M. unsat-pre M ∧ to-AF bot ∈ N
}›
    then show ?thesis
      by fastforce
  qed
```

qed

The following proof works as follows.

We assume that $(Inf, (Red_I, Red_F))$ is statically complete. From that and theorem *Calculi-And-Annotations.statically-complete-calculus bot FInf* $(\models)$ $FRed_I$ $FRed_F \Longrightarrow$ *Calculi-And-Annotations.statically-complete-calculus* (*to-AF bot*) *SInf* $(\models_{AF})$ $SRed_I$ $SRed_F$, we obtain that $(SInf, (SRed_I, SRed_F))$ is statically complete. This means that for all $\mathcal{N} \subseteq UNIV$, if $\mathcal{N}$ is saturated w.r.t. $(SInf, SRed_I)$ and $\mathcal{N} \models\cup_{AF} \{\bot\}$ then $\bot \in \mathcal{N}$. Since $\models\cup_{AF} \equiv \models\cap_{AF}$ when the right hand side is a singleton set, we have that for all $\mathcal{N} \subseteq UNIV$, if $\mathcal{N}$ is saturated w.r.t. $(SInf, SRed_I)$ and $\mathcal{N} \models\cap_{AF} \{\bot\}$ then $\bot \in \mathcal{N}$.

Because $\models\cap_{AF}$ is a consequence relation for the Saturation Framework, we can derive that $(SInf, (SRed_I, SRed_F))$ is dynamically complete (using the conjunctive entailment). We then proceed as above but in the opposite way to show that $(SInf, (SRed_I, SRed_F))$ is dynamically complete using the disjunctive entailment $\models\cup_{AF}$.

corollary *S-calculus-dynamically-complete*:
assumes *F-statically-complete*: ‹*statically-complete-calculus bot FInf* $(\models)$ $FRed_I$ $FRed_F$›
shows ‹*dynamically-complete-calculus* (*to-AF bot*) *SInf* $(\models_{AF})$ $SRed_I$ $SRed_F$›
proof −
have ‹*statically-complete-calculus* (*to-AF bot*) *SInf* $(\models_{AF})$ $SRed_I$ $SRed_F$›
using *S-calculus-statically-complete F-statically-complete*
by *blast*
then have ‹*statically-complete-calculus-axioms* (*to-AF bot*) *SInf* $(\models\cap_{AF})$ $SRed_I$›
using *entails-conj-is-entails-disj-if-right-singleton*[**where** *?C* = ‹*to-AF bot*›]
unfolding *statically-complete-calculus-def statically-complete-calculus-axioms-def*
by *blast*
then have ‹*Calculus.statically-complete-calculus-axioms* {*to-AF bot*} *SInf* $(\models\cap_{AF})$ $SRed_I$›
unfolding *statically-complete-calculus-axioms-def*
Calculus.statically-complete-calculus-axioms-def
using *saturated-equiv*
by *blast*
then have ‹*Calculus.statically-complete-calculus* {*to-AF bot*} *SInf* $(\models\cap_{AF})$ $SRed_I$ $SRed_F$›
using *Calculus.statically-complete-calculus.intro S-with-conj-is-calculus*
by *blast*
then have ‹*Calculus.dynamically-complete-calculus* {*to-AF bot*} *SInf* $(\models\cap_{AF})$ $SRed_I$ $SRed_F$›
using *S-with-conj-is-calculus calculus.dyn-equiv-stat*
by *blast*
then have ‹*Calculus.dynamically-complete-calculus-axioms* {*to-AF bot*} *SInf* $(\models\cap_{AF})$ $SRed_I$ $SRed_F$›
using *Calculus.dynamically-complete-calculus-def*
by *blast*

then have ‹*dynamically-complete-calculus-axioms* (*to-AF bot*) *SInf* ($\models\cap_{AF}$) $SRed_I$ $SRed_F$›
unfolding *dynamically-complete-calculus-axioms-def*
Calculus.dynamically-complete-calculus-axioms-def
by (*metis derivation-equiv fair-equiv llhd.rep-eq llnth.rep-eq singletonD singletonI*)
then have ‹*dynamically-complete-calculus-axioms* (*to-AF bot*) *SInf* ($\models_{AF}$) $SRed_I$ $SRed_F$›
unfolding *dynamically-complete-calculus-axioms-def*
using *entails-conj-is-entails-disj-if-right-singleton*
by *presburger*
then show ‹*dynamically-complete-calculus* (*to-AF bot*) *SInf* ($\models_{AF}$) $SRed_I$ $SRed_F$›
by (*simp add*: *dynamically-complete-calculus-def*
S-calculus.calculus-axioms)
qed

8.4 Strong completeness

theorem *S-calculus-strong-statically-complete*:
assumes *F-statically-complete*: ‹*statically-complete-calculus bot FInf* ($\models$) $FRed_I$ $FRed_F$› **and**
$\mathcal{N}$*-locally-saturated*: ‹*locally-saturated* $\mathcal{N}$› **and**
$\mathcal{N}$*-entails-bot*: ‹$\mathcal{N} \models_{AF} \{$*to-AF bot*$\}$›
shows ‹*to-AF bot* $\in \mathcal{N}$›
using $\mathcal{N}$*-locally-saturated*
unfolding *locally-saturated-def*
proof (*elim disjE*)
show ‹*to-AF bot* $\in \mathcal{N} \Longrightarrow$ *to-AF bot* $\in \mathcal{N}$›
by *blast*
next
assume ‹$\exists\ J.\ J \models_p \mathcal{N} \wedge$ *saturated* ($\mathcal{N}$ $proj_J$ J)›
then obtain J **where** J*-prop-model-of-*$\mathcal{N}$: ‹$J \models_p \mathcal{N}$› **and**
$\mathcal{N}$*-proj-J-saturated*: ‹*saturated* ($\mathcal{N}$ $proj_J$ J)›
by *blast*
then have ‹$\mathcal{N}$ $proj_J$ $J \models \{$*bot*$\}$›
using $\mathcal{N}$*-entails-bot AF-entails-def enabled-to-AF-set*
by (*metis* (*no-types*, *lifting*) *f-of-to-AF image-insert image-is-empty*)
then have ‹*bot* $\in \mathcal{N}$ $proj_J$ J›
using $\mathcal{N}$*-proj-J-saturated F-statically-complete*
by (*simp add*: *statically-complete-calculus.statically-complete*)
then show ‹*to-AF bot* $\in \mathcal{N}$›
using J*-prop-model-of-*$\mathcal{N}$
using *enabled-projection-def propositional-model-def propositional-projection-def*
by *force*
qed

lemma *locally-fair-derivation-is-saturated-at-liminf*:
‹*is-derivation S-calculus.derive* $\mathcal{N}i \Longrightarrow$ *locally-fair* $\mathcal{N}i \Longrightarrow$ *locally-saturated* (*lim-inf* $\mathcal{N}i$)›
proof −
assume $\mathcal{N}i$-*is-derivation*: ‹*is-derivation S-calculus.derive* $\mathcal{N}i$› **and**
‹*locally-fair* $\mathcal{N}i$›
then show ‹*locally-saturated* (*lim-inf* $\mathcal{N}i$)›
unfolding *locally-fair-def*
proof (*elim disjE*)
assume ‹$\exists$ *i*. *to-AF bot* ∈ *llnth* $\mathcal{N}i$ *i*›
then obtain *i* **where** ‹*to-AF bot* ∈ *llnth* $\mathcal{N}i$ *i*›
by *blast*
then have ‹*to-AF bot* ∈ *lim-inf* $\mathcal{N}i$›
using $\mathcal{N}$*i-is-derivation*[*OF* $\mathcal{N}$*i-is-derivation*]
by *blast*
then show *?thesis*
unfolding *locally-saturated-def*
by *blast*
next
assume ‹$\exists$ *J*. *J* $\models_p$ *limit* $\mathcal{N}i$ ∧ *Inf-from* (*limit* $\mathcal{N}i$ $proj_J$ *J*) ⊆ ($\bigcup$ *i*. $FRed_I$ (*llnth* $\mathcal{N}i$ *i* $proj_J$ *J*))›
then obtain *J* **where** *J-prop-model-of-limit*: ‹*J* $\models_p$ *limit* $\mathcal{N}i$› **and**
all-inf-of-limit-are-redundant:
‹*Inf-from* (*limit* $\mathcal{N}i$ $proj_J$ *J*) ⊆ ($\bigcup$ *i*. $FRed_I$ (*llnth* $\mathcal{N}i$ *i* $proj_J$ *J*))›
by *blast*
then have ‹$\forall$ *i*. *llnth* $\mathcal{N}i$ *i* ⊆ *lim-inf* $\mathcal{N}i$ ∪ $SRed_F$ (*lim-inf* $\mathcal{N}i$)›
using *Calculus.calculus.i-in-Liminf-or-Red-F*[*OF S-with-conj-is-calculus*, *of* ‹*to-llist* $\mathcal{N}i$›]
derivation-equiv[*of* ‹$\mathcal{N}i$›]
by (*simp add*: *Liminf-infinite-llist.rep-eq* $\mathcal{N}$*i-is-derivation llength-of-to-llist-is-infinite llnth.rep-eq sup-commute*)
then have ‹$\forall$ *i*. *llnth* $\mathcal{N}i$ *i* $proj_J$ *J* ⊆ (*lim-inf* $\mathcal{N}i$ $proj_J$ *J*) ∪ $FRed_F$ (*lim-inf* $\mathcal{N}i$ $proj_J$ *J*)›
by (*smt* (*verit*, *best*) *SRed-of-lim-inf UN-iff UnE UnI1 Un-commute Union-of-enabled-projection-is-enabled-projection subset-iff*)
then have $FRed_I$-*in-Red-I-of-*$FRed_F$:
‹($\bigcup$ *i*. $FRed_I$ (*llnth* $\mathcal{N}i$ *i* $proj_J$ *J*)) ⊆
($\bigcup$ *i*. $FRed_I$ ((*lim-inf* $\mathcal{N}i$ $proj_J$ *J*) ∪ $FRed_F$ (*lim-inf* $\mathcal{N}i$ $proj_J$ *J*)))›
by (*meson Red-I-of-subset SUP-mono UNIV-I*)
then have ‹($\bigcup$ *i*. $FRed_I$ (*llnth* $\mathcal{N}i$ *i* $proj_J$ *J*)) ⊆ ($\bigcup$ *i*. $FRed_I$ (*lim-inf* $\mathcal{N}i$ $proj_J$ *J*))›
using *Red-I-of-inf-*$FRed_F$*-subset-Red-I-of-inf*
by *auto*
then show *?thesis*
unfolding *locally-saturated-def*
using *J-prop-model-of-limit all-inf-of-limit-are-redundant saturated-def*
by *force*
qed

qed

theorem *S-calculus-strong-dynamically-complete*:
assumes *F-statically-complete*: ‹*statically-complete-calculus bot FInf* (⊨) $FRed_I$ $FRed_F$› **and**
𝒩i-is-derivation: ‹*is-derivation S-calculus.derive 𝒩i*› **and**
𝒩i-is-locally-fair: ‹*locally-fair 𝒩i*› **and**
𝒩i0-entails-bot: ‹*llhd 𝒩i* $\models_{AF}$ {*to-AF bot*}›
shows ‹∃ *i*. *to-AF bot* ∈ *llnth 𝒩i i*›
proof −
have ‹*llhd 𝒩i* ⊆ (⋃ *i*. *llnth 𝒩i i*)›
by (*simp add*: *SUP-upper llhd-is-llnth-0*)
then have ‹(⋃ *i*. *llnth 𝒩i i*) $\models_{AF}$ {*to-AF bot*}›
using *𝒩i0-entails-bot*
by (*meson AF-cons-rel.entails-trans AF-cons-rel.subset-entailed*
entails-conj-is-entails-disj-if-right-singleton)
then have ‹(⋃ *i*. *llnth 𝒩i i*) − $SRed_F$ (⋃ *i*. *llnth 𝒩i i*) $\models_{AF}$ {*to-AF bot*}›
using $SRed_F$*-entails-bot*
by *blast*
moreover have ‹*chain* (*Calculus.calculus.derive* $SRed_F$) (*to-llist 𝒩i*)›
using *derivation-equiv*[*of* ‹*𝒩i*›] *𝒩i-is-derivation*
by *blast*
then have ‹*Sup-llist* (*to-llist 𝒩i*) − *Liminf-llist* (*to-llist 𝒩i*) ⊆ $SRed_F$ (*Sup-llist* (*to-llist 𝒩i*))›
using *Calculus.calculus.Red-in-Sup*[*OF S-with-conj-is-calculus*]
by *blast*
then have ‹(⋃ *i*. *llnth 𝒩i i*) − $SRed_F$ (⋃ *i*. *llnth 𝒩i i*) ⊆ *lim-inf 𝒩i*›
by (*transfer fixing*: $FRed_F$, *unfold Sup-llist-def Liminf-llist-def*, *auto*)
ultimately have *𝒩i-inf-entails-bot*: ‹*lim-inf 𝒩i* $\models_{AF}$ {*to-AF bot*}›
by (*meson AF-cons-rel.entails-subsets subset-iff*)
then have *𝒩i-inf-locally-saturated*: ‹*locally-saturated* (*lim-inf 𝒩i*)›
using *𝒩i-is-derivation 𝒩i-is-locally-fair*
using *locally-fair-derivation-is-saturated-at-liminf*
by *blast*
then have ‹*to-AF bot* ∈ *lim-inf 𝒩i*›
using *F-statically-complete S-calculus-strong-statically-complete 𝒩i-inf-entails-bot*
by *blast*
then show ‹∃ *i*. *to-AF bot* ∈ *llnth 𝒩i i*›
by (*transfer fixing*: *bot*)
(*meson Liminf-llist-imp-exists-index*)
qed

end

9 Extensions: Inferences and simplifications

9.1 Simplifications

datatype *'f simplification* =
  Simplify (*S-from*: ‹*'f set*›) (*S-to*: ‹*'f set*›)

Simplification rules are said to be sound if every conclusion is entailed by all premises. We could have also used our conjunctive entailment *consequence-relation.entails-conjunctive*, but it is defined that way so there is nothing to worry about.

locale *AF-calculus* = *sc*: *sound-calculus bot Inf entails entails-sound* Red_I Red_F
  for
    bot :: (*'f*, *'v* :: *countable*) *AF* **and**
    Inf :: ‹(*'f*, *'v*) *AF inference set*› **and**
    entails :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool* **and**
    entails-sound :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool* **and**
    Red_I :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF inference set* **and**
    Red_F :: (*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set*

locale *AF-calculus-extended* =
  AF-calculus bot Inf entails entails-sound Red_I Red_F
  for *bot* :: ‹(*'f*, *'v* :: *countable*) *AF*› **and**
      Inf :: ‹(*'f*, *'v*) *AF inference set*› **and**
      entails :: ‹(*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool*› **and**
      entails-sound :: ‹(*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s$› *50*)
and
      Red_I :: ‹(*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF inference set*› **and**
      Red_F :: ‹(*'f*, *'v*) *AF set* ⇒ (*'f*, *'v*) *AF set*›
  + **fixes**
      Simps :: ‹(*'f*, *'v*) *AF simplification set*› **and**
      OptInfs :: ‹(*'f*, *'v*) *AF inference set*›
    assumes
      simps-simp: ‹$\delta \in Simps \Longrightarrow$ (*S-from* δ − *S-to* δ) ⊆ Red_F (*S-to* δ)› **and**
      simps-sound: ‹$\delta \in Simps \Longrightarrow \forall \mathcal{C} \in$ *S-to* δ. *S-from* $\delta \models s \{\mathcal{C}\}$› **and**

      infs-sound: ‹$\iota \in OptInfs \Longrightarrow$ *set* (*prems-of* ι) $\models s$ {*concl-of* ι}›
begin

lemma *simp-in-derivations*: ‹$\delta \in Simps \Longrightarrow$
  sc.derive ($\mathcal{M}$ ∪ *S-from* δ) ($\mathcal{M}$ ∪ *S-to* δ)›
  unfolding *sc.derive-def*
proof
  fix *C*
  assume *d-in*: ‹$\delta \in Simps$› **and**
    ‹$C \in \mathcal{M}$ ∪ *S-from* δ − ($\mathcal{M}$ ∪ *S-to* δ)›
  then have ‹$C \in$ *S-from* δ − *S-to* δ›
    by *blast*
  then show ‹$C \in Red_F$ ($\mathcal{M}$ ∪ *S-to* δ)›
   using *simps-simp*[*OF d-in*] **by** (*meson sc.Red-F-of-subset subset-eq sup.cobounded2*)

qed

lemma *opt-infs-in-derivations*: ‹$\iota \in$ *OptInfs* $\Longrightarrow$
sc.derive ($\mathcal{M} \cup$ *set* (*prems-of* ι)) ($\mathcal{M} \cup$ *set* (*prems-of* ι) $\cup$ {*concl-of* ι})›
unfolding *sc.derive-def*
proof
fix *C*
assume ‹$\iota \in$ *OptInfs*› **and**
C-in: ‹$C \in \mathcal{M} \cup$ *set* (*prems-of* ι) $-$ ($\mathcal{M} \cup$ *set* (*prems-of* ι) $\cup$ {*concl-of* ι})›
have ‹$\mathcal{M} \cup$ *set* (*prems-of* ι) $-$ ($\mathcal{M} \cup$ *set* (*prems-of* ι) $\cup$ {*concl-of* ι}) = {}›
by *blast*
then have *False* **using** *C-in* **by** *auto*
then show ‹$C \in$ *Red*$_F$ ($\mathcal{M} \cup$ *set* (*prems-of* ι) $\cup$ {*concl-of* ι})›
by *auto*
qed

end

Empty sets of simplifications and optional inferences are accepted in term*locale AF-calculus-extended*

context *AF-calculus*
begin

sublocale *empty-simps*:
AF-calculus-extended bot Inf entails entails-sound Red$_I$ *Red*$_F$
{} :: ('*f*, '*v*) *AF simplification set* {} :: ('*f*, '*v*) *AF inference set*
by (*unfold-locales*, *auto*)

end

Here we extend our basic calculus with simplification rules, one at a time:

- Split performs a n-ary case analysis on the head of the premise;
- Collect performs garbage collection on clauses which contain propositionally unsatisfiable heads;
- Trim removes assertions which are entailed by others.

9.1.1 The Split Rule

locale *AF-calculus-with-split* =
base-calculus: *AF-calculus-extended bot SInf entails entails-sound SRed*$_I$
SRed$_F$ *Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* $\Rightarrow$ ('*f*, '*v*) *AF set* $\Rightarrow$ *bool*› (**infix** ‹$\models_{AF}$› *50*) **and**
entails-sound :: ‹('*f*, '*v*) *AF set* $\Rightarrow$ ('*f*, '*v*) *AF set* $\Rightarrow$ *bool*› (**infix** ‹$\models_{AF}s$› *50*)
and

$SRed_I$ :: ‹('f, 'v) AF set ⇒ ('f, 'v) AF inference set› **and**
$SRed_F$ :: ‹('f, 'v) AF set ⇒ ('f, 'v) AF set› **and**
Simps :: ‹('f, 'v) AF simplification set› **and**
OptInfs :: ‹('f, 'v) AF inference set›
+ **fixes**
splittable :: ‹('f, 'v) AF ⇒ ('f, 'v) AF fset ⇒ bool›
assumes

split-sound1: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ⟹
{$\mathcal{C}$} $\models_{AF}s$ {*AF.Pair* (*F-of bot*) (*ffUnion* (*fimage neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$)}› **and**
split-sound2: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ⟹ ∀ $\mathcal{C}'$ ∈ *fset* $\mathcal{C}s$. {$\mathcal{C}$} $\models_{AF}s$ {$\mathcal{C}'$}› **and**
split-simp: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ⟹ $\mathcal{C}$ ∈ $SRed_F$
({*AF.Pair* (*F-of bot*) (*ffUnion* ((|‘|) *neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$)} ∪ *fset* $\mathcal{C}s$)›
begin

Rule definitions follow a similar naming convention to our two inference rules Base and Unsat defined in *annotated-calculus*: *X_simp* is the definition of the simplification rule, while *X_pre* is some precondition which must hold for the rule to be applicable.

abbreviation *split-pre* :: ‹('f, 'v) AF ⇒ ('f, 'v) AF fset ⇒ bool› **where**
‹*split-pre* $\mathcal{C}$ $\mathcal{C}s$ ≡ *splittable* $\mathcal{C}$ $\mathcal{C}s$›

abbreviation *split-res* :: ‹('f, 'v) AF ⇒ ('f, 'v) AF fset ⇒ ('f, 'v) AF set› **where**
‹*split-res* $\mathcal{C}$ $\mathcal{C}s$ ≡
(*insert* (*AF.Pair* (*F-of bot*) (*ffUnion* (*fimage neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$)) (*fset* $\mathcal{C}s$))›

abbreviation *split-simp* :: ‹('f, 'v) AF ⇒ ('f, 'v) AF fset ⇒ ('f, 'v) AF simplification› **where**
‹*split-simp* $\mathcal{C}$ $\mathcal{C}s$ ≡ *Simplify* {$\mathcal{C}$} (*split-res* $\mathcal{C}$ $\mathcal{C}s$)›

inductive-set *Simps-with-Split* :: ‹('f, 'v) AF simplification set› **where**
split: ‹*split-pre* $\mathcal{C}$ $\mathcal{C}s$ ⟹ *split-simp* $\mathcal{C}$ $\mathcal{C}s$ ∈ *Simps-with-Split*›
| *other*: ‹*simp* ∈ *Simps* ⟹ *simp* ∈ *Simps-with-Split*›

theorem *Inf-with-split-sound-wrt-entails-sound*:
‹ι ∈ *Simps-with-Split* ⟹ ∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι $\models_{AF}s$ {$\mathcal{C}$}›
proof −
assume ι*-is-simp-rule*: ‹ι ∈ *Simps-with-Split*›
then show ‹∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι $\models_{AF}s$ {$\mathcal{C}$}›
proof (*intro ballI*)
fix $\mathcal{C}$
assume $\mathcal{C}$*-is-consq-of-*ι: ‹$\mathcal{C}$ ∈ *S-to* ι›

```
    show ‹S-from ι ⊨_AF s {C}›
      using ι-is-simp-rule
    proof (cases rule: Simps-with-Split.cases)
      case (split C' Cs)

      have ‹S-from ι ⊨_AF s {AF.Pair (F-of bot) (ffUnion (fimage neg |‘| A-of |‘|
Cs) |∪| A-of C')}›
        using split-sound1 split by auto
      moreover have ‹∀C' ∈ fset Cs. S-from ι ⊨_AF s {C'}›
        using split-sound2 split by auto
      ultimately show ?thesis
        using C-is-consq-of-ι split(1) unfolding S-to-def by auto
    next
      case other
      then show ?thesis
        using C-is-consq-of-ι base-calculus.simps-sound by auto
    qed
  qed
qed


lemma split-redundant: ‹split-pre C Cs ⟹ C ∈ SRed_F (split-res C Cs)›
proof −
  assume pre-cond: ‹split-pre C Cs›
  then show ‹C ∈ SRed_F (split-res C Cs)›
    using split-simp by simp
qed


lemma simps-with-split-are-simps: ‹ι ∈ Simps-with-Split ⟹ (S-from ι − S-to ι)
⊆ SRed_F (S-to ι)›
proof
  fix C
  assume i-in: ‹ι ∈ Simps-with-Split› and
    C-in: ‹C ∈ S-from ι − S-to ι›
  then show ‹C ∈ SRed_F (S-to ι)›
  proof (cases rule: Simps-with-Split.cases)
    case (split C' Cs)
    then have ‹C = C'› using C-in by auto
    moreover have ‹S-to ι = split-res C' Cs› using split(1) simplification.sel(2)
by auto
    ultimately show ?thesis
      using split-redundant[OF split(2)] by presburger
  next
    case other
    then show ?thesis
      using base-calculus.simps-simp C-in by blast
  qed
qed
```

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails entails-sound* $SRed_I$ $SRed_F$ *Simps-with-Split OptInfs*
using *simps-with-split-are-simps Inf-with-split-sound-wrt-entails-sound base-calculus.infs-sound* **by** (*unfold-locales*, *auto*)

end

locale *splitting-calculus* =
core: *annotated-calculus bot Inf entails entails-sound* Red_I Red_F *fml asn*
for *bot* :: *'f* **and**
Inf :: ‹*'f inference set*› **and**
entails :: ‹*'f set* ⇒ *'f set* ⇒ *bool*› (**infix** ‹⊨› *50*) **and**
entails-sound :: ‹*'f set* ⇒ *'f set* ⇒ *bool*› (**infix** ‹⊨*s*› *50*) **and**
Red_I :: ‹*'f set* ⇒ *'f inference set*› **and**
Red_F :: ‹*'f set* ⇒ *'f set*› **and**
fml :: ‹*'v* :: *countable* ⇒ *'f*› **and**
asn :: ‹*'f sign* ⇒ (*'v* :: *countable*) *sign set*›
begin

interpretation *AF-sound-cons-rel*: *consequence-relation* ‹*to-AF bot*› ‹(⊨s_{AF})›
by (*rule core.AF-ext-sound-cons-rel*)

interpretation *SInf-sound-inf-system*: *sound-inference-system core.SInf* ‹*to-AF bot*› ‹(⊨s_{AF})›
by (*standard*, *auto simp add*: *core.SInf-sound-wrt-entails-sound*)

Rule definitions follow a similar naming convention to our two inference rules Base and Unsat defined in *annotated-calculus*: *X_simp* is the definition of the simplification rule, while *X_pre* is some precondition which must hold for the rule to be applicable.

definition *split-form* :: ‹*'f* ⇒ *'f fset* ⇒ *bool*› **where**
‹*split-form C Cs* ⟷ *C* ≠ *bot* ∧ *fcard Cs* ≥ *2*
∧ {*C*} ⊨*s fset Cs* ∧ (∀ *C'*. *C'* |∈| *Cs* ⟶ *C* ∈ Red_F {*C'*})›

definition *mk-split* :: ‹*'f* ⇒ *'f fset* ⇒ (*'f*, *'v*) *AF fset*› **where**
‹*split-form C Cs* ⟹ *mk-split C Cs* ≡ (λ *C'*. *AF.Pair C'* {| *SOME a*. *a* ∈ *asn* (*Pos C'*) |}) |‘| *Cs*›

definition *splittable* :: ‹(*'f*, *'v*) *AF* ⇒ (*'f*, *'v*) *AF fset* ⇒ *bool*› **where**
‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ≡ *split-form* (*F-of* $\mathcal{C}$) (*F-of* |‘| $\mathcal{C}s$) ∧ *mk-split* (*F-of* $\mathcal{C}$) (*F-of* |‘| $\mathcal{C}s$) = $\mathcal{C}s$›

lemma *split-creates-singleton-assertion-sets*:
‹*splittable* $\mathcal{C}$ $\mathcal{C}s$ ⟹ *A* |∈| $\mathcal{C}s$ ⟹ (∃ *a*. *A-of A* = {| *a* |})›
using *mk-split-def* **unfolding** *splittable-def* **by** (*metis* (*no-types*, *lifting*) *AF.sel*(*2*) *fimageE*)

lemma *split-all-assertion-sets-asn*:

‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow A$ |∈| $\mathcal{C}s \Longrightarrow (\exists\ a.\ A\text{-}of\ A = \{|a|\} \wedge a \in asn\ (Pos\ (F\text{-}of\ A)))$›
proof −
assume *pre-cond*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$› **and**
A-elem-As: ‹A |∈| $\mathcal{C}s$›
then have *pre-cond1*: ‹*split-form* (*F-of* $\mathcal{C}$) (*F-of* |‘| $\mathcal{C}s$)› **and**
pre-cond2: ‹*mk-split* (*F-of* $\mathcal{C}$) (*F-of* |‘| $\mathcal{C}s$) = $\mathcal{C}s$›
unfolding *splittable-def* **by** *auto*
have *mk-split-applied-def*: ‹*mk-split* (*F-of* $\mathcal{C}$) (*F-of* |‘| $\mathcal{C}s$) ≡
(λ C'. *AF.Pair* C' {| *SOME a*. $a \in asn\ (Pos\ C')$ |}) |‘| (*F-of* |‘| $\mathcal{C}s$)›
using *mk-split-def pre-cond* **unfolding** *splittable-def* **by** (*smt* (*verit*, *del-insts*))
have ‹∃ *a*. *A-of* $A = \{|a|\}$›
using *pre-cond A-elem-As* **by** (*simp add*: *split-creates-singleton-assertion-sets*)
then obtain a **where** *A-of-A-singleton-a*: ‹*A-of* $A = \{|a|\}$›
by *blast*
then have ‹$a \in asn\ (Pos\ (F\text{-}of\ A))$›
using *pre-cond2 A-elem-As core.asn-not-empty some-in-eq* **unfolding** *mk-split-applied-def*
by (*smt* (*z3*) *AF.exhaust-sel AF.inject FSet.fsingletonE fimage.rep-eq finsertI1 imageE someI-ex*)
then show ‹∃ *a*. *A-of* $A = \{|a|\} \wedge a \in asn\ (Pos\ (F\text{-}of\ A))$›
using *A-of-A-singleton-a* **by** *blast*
qed

lemma *split-all-pairs-in-As-in-Cs*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow (\forall\ P.\ P$ |∈| $\mathcal{C}s \longrightarrow$ *F-of* P |∈| (*F-of* |‘| $\mathcal{C}s$))›
using *mk-split-def*
by *fastforce*

lemma *split-all-pairs-in-Cs-in-As*:
‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow (\forall\ C.\ C$ |∈| (*F-of* |‘| $\mathcal{C}s$) $\longrightarrow (\exists\ a.\ AF.Pair\ C\ \{|a|\}$ |∈| $\mathcal{C}s))$›
using *mk-split-def* **unfolding** *splittable-def*
by *fastforce*

lemma *split-not-empty*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow \mathcal{C}s \neq \{||\}$›
unfolding *splittable-def split-form-def*
by (*metis bot-nat-0.extremum fcard-fempty fimage-fempty le-antisym nat.simps*(*3*) *numerals*(*2*))

notation *core.sound-cons.entails-neg* (**infix** ‹$\models s_\sim$› *50*)

lemma *split-sound1*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow$
$\{\mathcal{C}\} \models s_{AF}\ \{AF.Pair\ bot\ (ffUnion\ (fimage\ neg$ |‘| *A-of* |‘| $\mathcal{C}s)$ |∪| *A-of* $\mathcal{C})\}$›
proof −
assume *split-cond*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$›
then have *split-form*: ‹*split-form* (*F-of* $\mathcal{C}$) (*F-of* |‘| $\mathcal{C}s$)› **and**
split-mk: ‹*mk-split* (*F-of* $\mathcal{C}$) (*F-of* |‘| $\mathcal{C}s$) = $\mathcal{C}s$›
unfolding *splittable-def* **by** *auto*
define *Cs* **where** ‹*Cs* = *F-of* |‘| $\mathcal{C}s$›

have *Cs-not-empty*: ‹$Cs \neq \{||\}$›
using *split-cond split-not-empty* **unfolding** *Cs-def* **by** *blast*
then have $\mathcal{C}s$*-not-empty*: ‹$\mathcal{C}s \neq \{||\}$›
using *mk-split-def*[*of* ‹*F-of* $\mathcal{C}$› ‹*Cs*›] *split-mk split-form*
fimage-of-non-fempty-is-non-fempty[*OF Cs-not-empty*] **unfolding** *Cs-def* **by** *fastforce*
have ‹*fcard Cs* $\geq$ *1*›
by (*simp add*: *Cs-not-empty Suc-le-eq non-zero-fcard-of-non-empty-set*)
then have *card-fset-Cs-ge-1*: ‹*card* (*Pos* ‘ *fset Cs*) $\geq$ *1*›
by (*metis Cs-not-empty bot-fset.rep-eq card-eq-0-iff empty-is-image finite-fset finite-imageI*
fset-cong less-one linorder-not-le)
have ‹{*F-of* $\mathcal{C}$} $\models s$ *fset Cs*›
using *split-form* **unfolding** *Cs-def split-form-def* **by** *blast*
then have *F-of-*$\mathcal{C}$*-entails-Cs*: ‹{*Pos* (*F-of* $\mathcal{C}$)} $\models s_\sim$ *Pos* ‘ *fset Cs*›
unfolding *core.sound-cons.entails-neg-def*
by (*smt* (*verit*, *del-insts*) *UnCI imageI mem-Collect-eq singleton-conv*
core.sound-cons.entails-subsets subsetI)

have *finite-image-Pos-Cs*: ‹*finite* (*Pos* ‘ *fset Cs*)›
using *finite-fset* **by** *blast*

have *all-*C_i*-entail-bot*: ‹*fset* (*ffUnion* (*fimage neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$) $\subseteq$ *total-strip J*
$\Longrightarrow$ *AF.Pair* C_i {|a_i|} |∈| $\mathcal{C}s$ $\Longrightarrow$ (*core.fml-ext* ‘ *total-strip J*) ∪ {*Pos* C_i} $\models s_\sim$ {*Pos bot*}›
for *J* C_i a_i
proof −
fix *J* $\mathcal{C}_i$ a_i
assume ‹*fset* (*ffUnion* (*fimage neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$) $\subseteq$ *total-strip J*› **and**
Pair-$\mathcal{C}_i$*-*a_i*-in-As*: ‹*AF.Pair* $\mathcal{C}_i$ {|a_i|} |∈| $\mathcal{C}s$›
then have ‹*neg* a_i ∈ *total-strip J*›
using *mk-disjoint-finsert*
by *fastforce*
then have *neg-fml-*a_i*-in-J*: ‹*neg* (*core.fml-ext* a_i) ∈ *core.fml-ext* ‘ *total-strip J*›
by (*metis core.fml-ext.simps*(*1*) *core.fml-ext.simps*(*2*) *image-iff is-Neg-to-V is-Pos-to-V*
neg.simps(*1*) *neg.simps*(*2*))
moreover have *a*$_i$*-in-asn-*$\mathcal{C}_i$: ‹a_i ∈ *asn* (*Pos* $\mathcal{C}_i$)›
using *split-all-assertion-sets-asn*[*OF split-cond Pair-*$\mathcal{C}_i$*-*a_i*-in-As*]
by *auto*
moreover have ‹{*Pos* $\mathcal{C}_i$} $\models s_\sim$ {*Pos* $\mathcal{C}_i$}›
by (*meson consequence-relation.entails-reflexive core.sound-cons.ext-cons-rel*)
then have ‹(*core.fml-ext* ‘ (*total-strip J* − {*neg* a_i}) ∪ {*Pos* $\mathcal{C}_i$}) $\models s_\sim$ {*Pos* $\mathcal{C}_i$, *Pos bot*}›
by (*smt* (*verit*, *best*) *Un-upper2 consequence-relation.entails-subsets insert-is-Un*
core.sound-cons.ext-cons-rel sup-ge1)
ultimately show ‹*core.sound-cons.entails-neg* ((*core.fml-ext* ‘ *total-strip J*) ∪

$\{Pos\ \mathcal{C}_i\})\ \{Pos\ bot\}\rangle$
proof −
have ‹(*core.fml-ext* ‘ *total-strip J* ∪ {*core.fml-ext* a_i}) $\models s_\sim$ ({*Pos bot*} ∪ {})›
by (*smt* (*z3*) *Bex-def-raw UnCI Un-commute Un-insert-right Un-upper2 neg-fml-*a_i*-in-J*
consequence-relation.entails-subsets insert-is-Un insert-subset core.sound-cons.ext-cons-rel core.sound-cons.pos-neg-entails-bot)
then show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *core.C-entails-fml Un-commute* a_i*-in-asn-*$\mathcal{C}_i$ *consequence-relation.entails-cut core.fml-ext-is-mapping insert-is-Un core.sound-cons.ext-cons-rel*)
qed
qed
then have ‹*fset* (*ffUnion* (*fimage neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$) ⊆ *total-strip J* ⟹
((*core.fml-ext* ‘ *total-strip J*) ∪ {*Pos* (*F-of* $\mathcal{C}$)}) $\models s_\sim$ {*Pos bot*}› **for** *J*
unfolding *splittable-def*
proof −
fix *J*
assume ‹*fset* (*ffUnion* (*fimage neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$) ⊆ *total-strip J*›
then have C_i*-head-of-pair-entails-bot*:
‹*AF.Pair* C_i {|a_i|} |∈| $\mathcal{C}s$ ⟹ (*core.fml-ext* ‘ *total-strip J*) ∪ {*Pos* C_i} $\models s_\sim$ {*Pos bot*}›
for C_i a_i
using *all-*C_i*-entail-bot*
by *blast*
then have ‹C_i |∈| *Cs* ⟹ (*core.fml-ext* ‘ *total-strip J*) ∪ {*Pos* C_i} $\models s_\sim$ {*Pos bot*}›
for C_i
proof −
fix C_i
assume ‹C_i |∈| *Cs*›
then have ‹∃ a_i. *AF.Pair* C_i {|a_i|} |∈| $\mathcal{C}s$›
using *split-all-pairs-in-Cs-in-As*[*OF split-cond*] **unfolding** *Cs-def* **by** *presburger*
then obtain a_i **where** ‹*AF.Pair* C_i {|a_i|} |∈| $\mathcal{C}s$›
by *blast*
then show ‹(*core.fml-ext* ‘ *total-strip J*) ∪ {*Pos* C_i} $\models s_\sim$ {*Pos bot*}›
using C_i*-head-of-pair-entails-bot* **by** *blast*
qed
then show ‹(*core.fml-ext* ‘ *total-strip J*) ∪ {*Pos* (*F-of* $\mathcal{C}$)} $\models s_\sim$ {*Pos bot*}›
using *core.sound-cons.entails-of-entails-iff*[*OF F-of-*$\mathcal{C}$*-entails-Cs finite-image-Pos-Cs card-fset-Cs-ge-1*] **by** *blast*
qed
then have
‹*fset* (*ffUnion* (*fimage neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$) ⊆ *total-strip J* ⟹
((*core.fml-ext* ‘ *total-strip J*) ∪ *Pos* ‘ ({$\mathcal{C}$} *proj*$_J$ *J*)) $\models s_\sim$ {*Pos bot*}›
for *J*
using *split-cond* **by** (*simp add*: *core.enabled-def core.enabled-projection-def*)

then show ‹$\{\mathcal{C}\} \models s_{AF}$ $\{AF.Pair\ bot\ (ffUnion\ (fimage\ neg\ |`|\ A\text{-}of\ |`|\ \mathcal{C}s)\ |\cup|\ A\text{-}of\ \mathcal{C})\}$›
unfolding *core.AF-entails-sound-def* **using** *core.enabled-def core.enabled-set-def*
by *simp*
qed

lemma *split-sound2*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow \forall\ \mathcal{C}' \in fset\ \mathcal{C}s.\ \{\mathcal{C}\} \models s_{AF}\ \{\mathcal{C}'\}$›
proof
fix $\mathcal{C}'$
assume *split-cond*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$› **and** $\mathcal{C}'$*-in*: ‹$\mathcal{C}'$ $|\in|$ $\mathcal{C}s$›
have ‹C'' $|\in|$ $\mathcal{C}s \Longrightarrow$ *fset* (*A-of* C'') $\subseteq$ *total-strip* $J \Longrightarrow$
(*core.fml-ext* ‘ *total-strip* J) $\cup$ *Pos* ‘ ($\{\mathcal{C}\}$ $proj_J$ J) $\models s_\sim$ $\{Pos\ (F\text{-}of\ C'')\}$›
for J C''
proof −
fix J C''
assume C''*-in-As*: ‹C'' $|\in|$ $\mathcal{C}s$› **and**
A-of-C''*-subset-J*: ‹*fset* (*A-of* C'') $\subseteq$ *total-strip* J›
then have ‹$\exists\ a_i.\ a_i \in asn\ (Pos\ (F\text{-}of\ C'')) \wedge A\text{-}of\ C'' = \{|\ a_i\ |\}$›
using *split-all-assertion-sets-asn*[*OF split-cond* C''*-in-As*] **by** *blast*
then obtain a_i **where** a_i*-in-asn-F-of-*C'': ‹$a_i \in asn\ (Pos\ (F\text{-}of\ C''))$› **and**
A-of-C''*-is*: ‹*A-of* $C'' = \{|\ a_i\ |\}$›
by *blast*
then show ‹(*core.fml-ext* ‘ *total-strip* J) $\cup$ *Pos* ‘ ($\{\mathcal{C}\}$ $proj_J$ J) $\models s_\sim$ $\{Pos\ (F\text{-}of\ C'')\}$›
by (*smt* (*verit*, *best*) *A-of-*C''*-subset-J consequence-relation.entails-subsets empty-subsetI*
finsert.rep-eq core.fml-entails-C core.fml-ext-is-mapping image-eqI insert-is-Un
insert-subset core.sound-cons.ext-cons-rel sup-ge1)
qed
then have *unfolded-AF-sound-entails*: ‹C'' $\in$ *fset* $\mathcal{C}s \Longrightarrow$ *fset* (*A-of* C'') $\subseteq$ *total-strip* $J \Longrightarrow$
(*core.fml-ext* ‘ *total-strip* J) $\cup$ *Pos* ‘ ($\{\mathcal{C}\}$ $proj_J$ J) $\models s_\sim$ $\{Pos\ (F\text{-}of\ C'')\}$›
for J C''
by *fast*

show ‹$\{\mathcal{C}\} \models s_{AF}$ $\{\mathcal{C}'\}$›
unfolding *core.AF-entails-sound-def core.enabled-set-def core.enabled-def*
using *unfolded-AF-sound-entails*[*OF* $\mathcal{C}'$*-in*] *split-cond* $\mathcal{C}'$*-in* **by** *auto*
qed

lemma *split-simp*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow \mathcal{C} \in$
core.SRed$_F$ ({ *AF.Pair bot* (*ffUnion* ((|`|) *neg* |`| *A-of* |`| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$) } ∪ *fset* $\mathcal{C}s$)›
proof −
assume *split-cond*: ‹*splittable* $\mathcal{C}$ $\mathcal{C}s$›
define *Cs* **where** ‹*Cs* = *F-of* |`| $\mathcal{C}s$›
then have *F-of-*$\mathcal{C}$*-not-bot*: ‹*F-of* $\mathcal{C} \neq$ *bot*› **and**

```
    ‹fcard Cs ≥ 2› and
    ‹{F-of C} |=s fset Cs› and
    C-red-to-splitted-Cs: ‹∀ C'. C' |∈| Cs ⟶ F-of C ∈ Red_F {C'}›
    using split-cond unfolding splittable-def split-form-def
    by blast+
  then have ‹∀ J. core.enabled C J ⟶
    F-of C ∈ Red_F (({ AF.Pair bot (ffUnion (fimage neg |‘| A-of |‘| Cs) |∪| A-of
C) } proj_J J)
    ∪ (fset Cs proj_J J))›
  proof (intro allI impI)
    fix J
    assume C-enabled: ‹core.enabled C J›
    then show
      ‹F-of C ∈ Red_F (({ AF.Pair bot (ffUnion (fimage neg |‘| A-of |‘| Cs) |∪| A-of
C) } proj_J J)
       ∪ (fset Cs proj_J J))›
    proof (cases ‹∃ A. A |∈| A-of |‘| Cs ∧ (∃ a. a |∈| A ∧ a ∈ total-strip J)›)
      case True
      then have ex-C-enabled-in-As: ‹∃ C. C |∈| Cs ∧ core.enabled C J›
        using core.enabled-def split-creates-singleton-assertion-sets split-cond
        by fastforce
      then have ‹∃ C. C ∈ fset Cs proj_J J›
        by (simp add: core.enabled-projection-def)
      then show ?thesis
        using C-red-to-splitted-Cs split-cond core.Red-F-of-subset[of ‹fset Cs proj_J
J›]
        mk-split-def by (smt (z3) CollectD Cs-def basic-trans-rules(31) core.enabled-projection-def
        core.sound-calculus-axioms insert-subset le-sup-iff sound-calculus.Red-F-of-subset
          split-all-pairs-in-As-in-Cs sup-bot.right-neutral sup-ge1)
    next
      case False
      then have ‹fset Cs proj_J J = {}›
        using split-creates-singleton-assertion-sets[OF split-cond]
      by (smt (verit, del-insts) Collect-empty-eq core.enabled-def core.enabled-projection-def
          fimage-finsert finsert.rep-eq finsertCI insert-subset mk-disjoint-finsert)
     moreover have ‹∀ A. A |∈| A-of |‘| Cs ⟶ (∀ a. a |∈| A ⟶ ¬ a ∈ total-strip
J)›
        using False
        by blast
      then have ‹∀ A. A |∈| A-of |‘| Cs ⟶ (∀ a. a |∈| A ⟶ neg a ∈ total-strip
J)›
        by auto
      then have ‹fset (ffUnion ((fimage neg ∘ A-of) |‘| Cs)) ⊆ total-strip J›
      by (smt (verit, best) fimage-iff fset.map-comp fset-ffUnion-subset-iff-all-fsets-subset
subsetI)
     then have ‹fset (ffUnion ((fimage neg ∘ A-of) |‘| Cs) |∪| A-of C) ⊆ total-strip
J›
        using C-enabled
        by (simp add: core.enabled-def)
```

then have ‹{*AF.Pair bot* (*ffUnion* ((*fimage neg* ∘ *A-of*) |`| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$)} $proj_J$ J = {*bot*}›
unfolding *core.enabled-projection-def core.enabled-def*
by *auto*
ultimately show *?thesis*
by (*simp add*: *F-of-C-not-bot core.all-red-to-bot*)
qed
qed
then show ‹$\mathcal{C}$ ∈ *core.SRed*$_F$ ({ *AF.Pair bot* (*ffUnion* ((|`|) *neg* |`| *A-of* |`| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$) } ∪ *fset* $\mathcal{C}s$)›
unfolding *core.SRed*$_F$*-def core.enabled-def*
by (*intro UnI1*) (*smt* (*verit*, *ccfv-threshold*) *AF.collapse CollectI core.distrib-proj*)
qed

sublocale *empty-simps*: *AF-calculus-extended to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ {} {}
proof
show ‹*core.SRed*$_I$ N ⊆ *core.SInf*› **for** N
using *core.SRed*$_I$*-in-SInf* **.**
next
show ‹N $\models_{AF}$ {*to-AF bot*} ⟹ N − *core.SRed*$_F$ N $\models_{AF}$ {*to-AF bot*}› **for** N
using *core.SRed*$_F$*-entails-bot* **.**
next
show ‹N ⊆ N' ⟹ *core.SRed*$_F$ N ⊆ *core.SRed*$_F$ N'› **for** N N'
using *core.SRed*$_F$*-of-subset-F* **.**
next
show ‹N ⊆ N' ⟹ *core.SRed*$_I$ N ⊆ *core.SRed*$_I$ N'› **for** N N'
using *core.SRed*$_I$*-of-subset-F* **.**
next
show ‹N' ⊆ *core.SRed*$_F$ N ⟹ *core.SRed*$_F$ N ⊆ *core.SRed*$_F$ (N − N')› **for** N N'
using *core.SRed*$_F$*-of-SRed*$_F$*-subset-F* **.**
next
show ‹N' ⊆ *core.SRed*$_F$ N ⟹ *core.SRed*$_I$ N ⊆ *core.SRed*$_I$ (N − N')› **for** N N'
using *core.SRed*$_I$*-of-SRed*$_F$*-subset-F* **.**
next
show ‹ι ∈ *core.SInf* ⟹ *concl-of* ι ∈ N ⟹ ι ∈ *core.SRed*$_I$ N› **for** ι N
using *core.S-calculus.Red-I-of-Inf-to-N* **.**
next
show ‹ι ∈ {} ⟹ *S-from* ι − *S-to* ι ⊆ *core.SRed*$_F$ (*S-to* ι)› **for** ι
by *simp*
next
show ‹ι ∈ {} ⟹ ∀$\mathcal{C}$∈*S-to* ι. *S-from* ι $\models s_{AF}$ {$\mathcal{C}$}› **for** ι
by *blast*
next
show ‹ι ∈ {} ⟹ *set* (*prems-of* ι) $\models s_{AF}$ {*concl-of* ι}› **for** ι
by *blast*
qed

lemma *extend-simps-with-split*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-split* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs splittable*›
proof −
interpret *sound-simps*: *AF-calculus-extended to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*
using *assms* **.**
show *?thesis*
proof
show ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow$ {$\mathcal{C}$} $\models s_{AF}$ {*AF.Pair* (*F-of* (*to-AF bot*)) (*ffUnion* ((|‘|) *neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$)}› **for** $\mathcal{C}$ $\mathcal{C}s$
using *split-sound1* **by** (*simp add*: *F-of-to-AF*)
next
show ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow \forall \mathcal{C}' \in |\mathcal{C}s.$ {$\mathcal{C}$} $\models s_{AF}$ {$\mathcal{C}'$}› **for** $\mathcal{C}$ $\mathcal{C}s$
using *split-sound2* **.**
next
show ‹*splittable* $\mathcal{C}$ $\mathcal{C}s \Longrightarrow \mathcal{C} \in$ *core.SRed*$_F$ ({*AF.Pair* (*F-of* (*to-AF bot*)) (*ffUnion* ((|‘|) *neg* |‘| *A-of* |‘| $\mathcal{C}s$) |∪| *A-of* $\mathcal{C}$)} ∪ *fset* $\mathcal{C}s$)›
for $\mathcal{C}$ $\mathcal{C}s$
using *split-simp* **by** (*simp add*: *F-of-to-AF*)
qed
qed

interpretation *splitting-calc-with-split*:
AF-calculus-with-split to-AF bot core.SInf ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ {} {} *splittable*
using *extend-simps-with-split*[*OF empty-simps.AF-calculus-extended-axioms*] **.**

end

9.1.2 The Collect Rule

locale *AF-calculus-with-collect* = *base-calculus*: *AF-calculus-extended bot SInf entails entails-sound SRed*$_I$ *SRed*$_F$ *Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models_{AF}$› *50*) **and**
entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*) **and**
SRed$_I$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
SRed$_F$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**

Simps :: ‹('f, 'v) *AF simplification set*› **and**
OptInfs :: ‹('f, 'v) *AF inference set*›
+ **assumes**
collect-redundant: ‹*F-of* $\mathcal{C} \neq$ *F-of bot* $\wedge$ $\mathcal{M} \models_{AF}$ {*AF.Pair* (*F-of bot*) (*A-of* $\mathcal{C}$)} $\wedge$
($\forall$ $\mathcal{C} \in \mathcal{M}$. *F-of* $\mathcal{C}$ = *F-of bot*) $\Longrightarrow$ $\mathcal{C} \in$ *SRed*$_F$ $\mathcal{M}$›
begin

interpretation *AF-sound-cons-rel*: *consequence-relation bot* ‹($\models s_{AF}$)›
using *base-calculus.sc.sound-cons.consequence-relation-axioms* **.**

interpretation *SInf-sound-inf-system*: *sound-inference-system SInf bot* ‹($\models s_{AF}$)›
using *base-calculus.sc.sound-inference-system-axioms* **.**

abbreviation *collect-pre* :: ‹('f, 'v) *AF* $\Rightarrow$ ('f, 'v) *AF set* $\Rightarrow$ *bool*› **where**
‹*collect-pre* $\mathcal{C}$ $\mathcal{M}$ $\equiv$
F-of $\mathcal{C} \neq$ *F-of bot* $\wedge$ $\mathcal{M} \models_{AF}$ {*AF.Pair* (*F-of bot*) (*A-of* $\mathcal{C}$)} $\wedge$ ($\forall$ $\mathcal{C} \in \mathcal{M}$. *F-of* $\mathcal{C}$ = *F-of bot*)›

abbreviation *collect-simp* :: ‹('f, 'v) *AF* $\Rightarrow$ ('f, 'v) *AF set* $\Rightarrow$ ('f, 'v) *AF simplification*› **where**
‹*collect-simp* $\mathcal{C}$ $\mathcal{M}$ $\equiv$ *Simplify* (*insert* $\mathcal{C}$ $\mathcal{M}$) $\mathcal{M}$›

inductive-set *Simps-with-Collect* :: ‹('f, 'v) *AF simplification set*› **where**
collect: ‹*collect-pre* $\mathcal{C}$ $\mathcal{M}$ $\Longrightarrow$ *collect-simp* $\mathcal{C}$ $\mathcal{M}$ $\in$ *Simps-with-Collect*›
| *other*: ‹$\iota \in$ *Simps* $\Longrightarrow$ $\iota \in$ *Simps-with-Collect*›

theorem *SInf-with-collect-sound-wrt-entails-sound*:
‹$\iota \in$ *Simps-with-Collect* $\Longrightarrow$ $\forall$ $\mathcal{C} \in$ *S-to* ι. *S-from* ι $\models s_{AF}$ {$\mathcal{C}$}›
proof −
assume ι*-is-simp-rule*: ‹$\iota \in$ *Simps-with-Collect*›
then show ‹$\forall$ $\mathcal{C} \in$ *S-to* ι. *S-from* ι $\models s_{AF}$ {$\mathcal{C}$}›
proof (*intro ballI*)
fix $\mathcal{C}$
assume $\mathcal{C}$*-is-consq-of-*ι: ‹$\mathcal{C} \in$ *S-to* ι›
show ‹*S-from* ι $\models s_{AF}$ {$\mathcal{C}$}›
using ι*-is-simp-rule*
proof (*cases rule*: *Simps-with-Collect.cases*)
case (*collect* $\mathcal{C}'$ $\mathcal{N}$)
then show *?thesis*
using $\mathcal{C}$*-is-consq-of-*ι **by** (*metis base-calculus.sc.sound-cons.entails-conjunctive-def*

base-calculus.sc.sound-cons.subset-entailed simplification.sel(*1*) *simplification.sel*(*2*)
subset-insertI)

```
    next
      case other
      then show ?thesis
        using base-calculus.simps-sound C-is-consq-of-ι by auto
    qed
  qed
qed


theorem simps-with-collect-are-simps:
  ‹ι ∈ Simps-with-Collect ⟹ (S-from ι − S-to ι) ⊆ SRed_F (S-to ι)›
proof
  fix C
  assume ι-is-simp-rule: ‹ι ∈ Simps-with-Collect› and
    C-in: ‹C ∈ S-from ι − S-to ι›
  then show ‹C ∈ SRed_F (S-to ι)›
  proof (cases rule: Simps-with-Collect.cases)
    case (collect C' M)
    then have ‹C = C'›
      using C-in by auto
    moreover have ‹S-to ι = M›
      using collect(1) by simp
    ultimately show ?thesis
      using collect-redundant[OF collect(2)] by blast
  next
    case other
    then show ?thesis
      using base-calculus.simps-simp C-in by blast
  qed
qed

sublocale AF-calc-ext: AF-calculus-extended bot SInf entails entails-sound SRed_I
SRed_F
  Simps-with-Collect OptInfs
  using simps-with-collect-are-simps SInf-with-collect-sound-wrt-entails-sound
    base-calculus.infs-sound by (unfold-locales, auto)

end

context splitting-calculus
begin


lemma collect-redundant: ‹F-of C ≠ bot ∧ M ⊨_AF {AF.Pair bot (A-of C)} ∧
      (∀ C ∈ M. F-of C = bot) ⟹ C ∈ core.SRed_F M›
proof −
  assume ‹F-of C ≠ bot ∧ M ⊨_AF {AF.Pair bot (A-of C)} ∧ (∀ C ∈ M. F-of C
= bot)›
  then have head-C-not-bot: ‹F-of C ≠ bot› and
```

```
    𝓜-entails-bot-𝓒: ‹𝓜 ⊨_AF {AF.Pair bot (A-of 𝓒)}› and
    all-heads-are-bot-in-𝓜: ‹∀ 𝓒 ∈ 𝓜. F-of 𝓒 = bot›
   by auto
  have ‹⋀ J. (∃ 𝓒' ∈ 𝓜. core.enabled 𝓒' J) ⟹ core.enabled 𝓒 J
        ⟹ F-of 𝓒 ∈ Red_F (𝓜 proj_J J)› and
      ‹⋀ J. ¬ (∃ 𝓒' ∈ 𝓜. core.enabled 𝓒' J) ⟹ core.enabled 𝓒 J ⟹ False›
  proof −
   fix J
   assume 𝓒-enabled: ‹core.enabled 𝓒 J› and
         ‹∃ 𝓒' ∈ 𝓜. core.enabled 𝓒' J›
   then have ‹𝓜 proj_J J = {bot}›
    using all-heads-are-bot-in-𝓜 unfolding core.enabled-projection-def by fast
   then show ‹F-of 𝓒 ∈ Red_F (𝓜 proj_J J)›
    using core.all-red-to-bot[OF head-𝓒-not-bot] by simp
  next
   fix J
   assume 𝓒-enabled: ‹core.enabled 𝓒 J› and
         ‹¬ (∃ 𝓒' ∈ 𝓜. core.enabled 𝓒' J)›
   then have 𝓜-proj-J-empty: ‹𝓜 proj_J J = {}›
    unfolding core.enabled-projection-def by blast
   moreover have ‹core.enabled (AF.Pair bot (A-of 𝓒)) J›
    using 𝓒-enabled by (auto simp add: core.enabled-def)
   ultimately have ‹{} ⊨ {bot}›
    using 𝓜-entails-bot-𝓒 using core.AF-entails-def by auto
   then show ‹False›
    using core.entails-bot-to-entails-empty core.entails-nontrivial by blast
  qed
  then show ‹𝓒 ∈ core.SRed_F 𝓜›
   unfolding core.SRed_F-def core.enabled-def by (smt (verit, ccfv-SIG) AF.collapse
CollectI UnI1)
qed


lemma extend-simps-with-collect:
  assumes
    ‹AF-calculus-extended (to-AF bot) core.SInf (⊨_AF) (⊨s_AF) core.SRed_I
     core.SRed_F Simps OptInfs›
  shows
    ‹AF-calculus-with-collect (to-AF bot) core.SInf (⊨_AF) (⊨s_AF) core.SRed_I
core.SRed_F Simps OptInfs›
proof −
  interpret sound-simps:
    AF-calculus-extended to-AF bot core.SInf (⊨_AF) (⊨s_AF) core.SRed_I
     core.SRed_F Simps
    using assms .
  show ?thesis
  proof
   fix 𝓒 𝓜
    show ‹F-of 𝓒 ≠ F-of (to-AF bot) ∧ 𝓜 ⊨_AF {AF.Pair (F-of (to-AF bot))
```

$(A\text{-}of\ \mathcal{C})\} \wedge$
$(\forall \mathcal{C} \in \mathcal{M}.\ F\text{-}of\ \mathcal{C} = F\text{-}of\ (to\text{-}AF\ bot)) \Longrightarrow \mathcal{C} \in core.SRed_F\ \mathcal{M}\rangle$
using *collect-redundant* **by** (*simp add*: *F-of-to-AF*)
qed
qed

interpretation *splitting-calc-with-collect*:
AF-calculus-with-collect to-AF bot core.SInf $(\models_{AF})$ $(\models s_{AF})$ *core.SRed*$_I$ *core.SRed*$_F$ {} {}
using *extend-simps-with-collect*[*OF empty-simps.AF-calculus-extended-axioms*] **.**

end

9.1.3 The Trim Rule

locale *AF-calculus-with-trim* = *base-calculus*: *AF-calculus-extended bot SInf entails entails-sound SRed*$_I$ *SRed*$_F$ *Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models_{AF}$› *50*) **and**
entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*)
and
SRed$_I$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
SRed$_F$ :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**
Simps :: ‹('*f*, '*v*) *AF simplification set*› **and**
OptInfs :: ‹('*f*, '*v*) *AF inference set*›
+ **assumes**
trim-sound: ‹*A-of* $\mathcal{C} = A \mid\cup\mid B \wedge$ *F-of* $\mathcal{C} \neq$ *F-of bot* $\wedge$
$\mathcal{M} \cup \{AF.Pair\ (F\text{-}of\ bot)\ A\} \models s_{AF} \{AF.Pair\ (F\text{-}of\ bot)\ B\} \wedge$
$(\forall \mathcal{C} \in \mathcal{M}.\ F\text{-}of\ \mathcal{C} = (F\text{-}of\ bot)) \wedge A \mid\cap\mid B = \{||\} \wedge A \neq \{||\}$
$\Longrightarrow$ *insert* $\mathcal{C}\ \mathcal{M} \models s_{AF} \{AF.Pair\ (F\text{-}of\ \mathcal{C})\ B\}$› **and**
trim-redundant: ‹*A-of* $\mathcal{C} = A \mid\cup\mid B \wedge$ *F-of* $\mathcal{C} \neq$ *F-of bot* $\wedge$
$\mathcal{M} \cup \{AF.Pair\ (F\text{-}of\ bot)\ A\} \models s_{AF} \{AF.Pair\ (F\text{-}of\ bot)\ B\} \wedge$
$(\forall \mathcal{C} \in \mathcal{M}.\ F\text{-}of\ \mathcal{C} = (F\text{-}of\ bot)) \wedge A \mid\cap\mid B = \{||\} \wedge A \neq \{||\}$
$\Longrightarrow \mathcal{C} \in SRed_F\ (\mathcal{M} \cup \{\ AF.Pair\ (F\text{-}of\ \mathcal{C})\ B\ \})$›

begin

interpretation *AF-sound-cons-rel*: *consequence-relation bot* ‹$(\models s_{AF})$›
using *base-calculus.sc.sound-cons.consequence-relation-axioms* **.**

interpretation *SInf-sound-inf-system*: *sound-inference-system SInf bot* ‹$(\models s_{AF})$›
using *base-calculus.sc.sound-inference-system-axioms* **.**

abbreviation *trim-pre* :: ‹('*f*, '*v*) *AF* ⇒ '*v sign fset* ⇒ '*v sign fset* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› **where**
‹*trim-pre* $\mathcal{C}$ *A B* $\mathcal{M} \equiv$ *A-of* $\mathcal{C} = A \mid\cup\mid B \wedge$ *F-of* $\mathcal{C} \neq$ *F-of bot* $\wedge$
$\mathcal{M} \cup \{\ AF.Pair\ (F\text{-}of\ bot)\ A\ \} \models s_{AF} \{\ AF.Pair\ (F\text{-}of\ bot)\ B\ \} \wedge$

$(\forall\ \mathcal{C} \in \mathcal{M}.\ \textit{F-of}\ \mathcal{C} = (\textit{F-of bot})) \wedge A\ |\cap|\ B = \{||\} \wedge A \neq \{||\}\rangle$

abbreviation *trim-simp* :: ‹('*f*, '*v*) *AF* ⇒ '*v sign fset* ⇒ '*v sign fset* ⇒ ('*f*, '*v*) *AF set* ⇒
('*f*, '*v*) *AF simplification*› **where**
‹*trim-simp* $\mathcal{C}$ *A B* $\mathcal{M}$ ≡ *Simplify* (*insert* $\mathcal{C}$ $\mathcal{M}$) (*insert* (*AF.Pair* (*F-of* $\mathcal{C}$) *B*) $\mathcal{M}$)›

inductive-set *Simps-with-Trim* :: ‹('*f*, '*v*) *AF simplification set*› **where**
trim: ‹*trim-pre* $\mathcal{C}$ *A B* $\mathcal{M}$ ⟹ *trim-simp* $\mathcal{C}$ *A B* $\mathcal{M}$ ∈ *Simps-with-Trim*›
| *other*: ‹ι ∈ *Simps* ⟹ ι ∈ *Simps-with-Trim*›

theorem *SInf-with-trim-sound-wrt-entails-sound*:
‹ι ∈ *Simps-with-Trim* ⟹ ∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι $\models s_{AF}$ {$\mathcal{C}$}›
proof −
assume *ι-is-simp-rule*: ‹ι ∈ *Simps-with-Trim*›
then show ‹∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι $\models s_{AF}$ {$\mathcal{C}$}›
proof (*intro ballI*)
fix $\mathcal{C}$
assume $\mathcal{C}$*-is-consq-of-ι*: ‹$\mathcal{C}$ ∈ *S-to* ι›
show ‹*S-from* ι $\models s_{AF}$ {$\mathcal{C}$}›
using *ι-is-simp-rule*
proof (*cases rule*: *Simps-with-Trim.cases*)
case (*trim* $\mathcal{C}'$ *A B* $\mathcal{N}$)
then have ‹*A-of* $\mathcal{C}'$ = *A* |∪| *B*› **and**
‹*F-of* $\mathcal{C}'$ ≠ *F-of bot*› **and**
$\mathcal{N}$*-and-Pair-bot-A-entails-Pair-bot-B*:
‹$\mathcal{N}$ ∪ {*AF.Pair* (*F-of bot*) *A*} $\models s_{AF}$ {*AF.Pair* (*F-of bot*) *B*}› **and**
all-heads-in-$\mathcal{N}$*-are-bot*: ‹(∀ $\mathcal{C}$ ∈ $\mathcal{N}$. *F-of* $\mathcal{C}$ = *F-of bot*)› **and**
‹*A* |∩| *B* = {||}› **and**
‹*A* ≠ {||}›
by *blast*+
consider ‹$\mathcal{C}$ = *AF.Pair* (*F-of* $\mathcal{C}'$) *B*› | ‹$\mathcal{C}$ ∈ $\mathcal{N}$›
using $\mathcal{C}$*-is-consq-of-ι trim*(*1*) **by** *auto*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
using *trim-sound*[*OF trim*(*2*)] *trim*(*1*) **by** *fastforce*
next
case *2*
then show *?thesis*
using *trim*(*1*) **by** (*metis base-calculus.sc.sound-cons.entails-reflexive*
base-calculus.sc.sound-cons.entails-subsets empty-subsetI insertI2 insert-subset
simplification.sel(*1*) *subset-singleton-iff*)

```
    qed
  next
    case other
    show ?thesis
      using base-calculus.simps-sound[OF other] C-is-consq-of-ι by auto
  qed
 qed
qed


theorem simps-with-trim-are-simps:
 ‹ι ∈ Simps-with-Trim ⟹ (S-from ι − S-to ι) ⊆ SRed_F (S-to ι)›
proof
 fix C
 assume ‹ι ∈ Simps-with-Trim› and
   C-in: ‹C ∈ S-from ι − S-to ι›
 then show ‹C ∈ SRed_F (S-to ι)›
 proof (cases rule: Simps-with-Trim.cases)
   case (trim C' A B M)
   then have ‹C' = C› using C-in by auto
   then show ?thesis
     using trim-redundant[OF trim(2)] trim(1) by simp
 next
   case other
   then show ?thesis
     using base-calculus.simps-simp C-in by auto
 qed
qed

sublocale AF-calc-ext: AF-calculus-extended bot SInf entails entails-sound SRed_I
SRed_F
 Simps-with-Trim OptInfs
 using simps-with-trim-are-simps SInf-with-trim-sound-wrt-entails-sound
   base-calculus.infs-sound by (unfold-locales, auto)

end

context splitting-calculus
begin

lemma trim-sound: ‹A-of C' = A |∪| B ∧ F-of C' ≠ bot ∧
     N ∪ {AF.Pair bot A} |=s_AF {AF.Pair bot B} ∧
     (∀C∈N. F-of C = bot) ∧ A |∩| B = {||} ∧ A ≠ {||}
       ⟹ C = AF.Pair (F-of C') B ⟹ insert C' N |=s_AF {C}›
proof −
 assume ‹A-of C' = A |∪| B ∧ F-of C' ≠ bot ∧
     N ∪ {AF.Pair bot A} |=s_AF {AF.Pair bot B} ∧
     (∀C∈N. F-of C = bot) ∧ A |∩| B = {||} ∧ A ≠ {||}› and
   C-is: ‹C = AF.Pair (F-of C') B›
```

```
  then have A-of-Cp: ‹A-of 𝒞' = A |∪| B› and
    F-of-Cp: ‹F-of 𝒞' ≠ bot› and
    A-in-N-entails-B: ‹AF.Pair bot A ▷ 𝒩 ⊨s_AF {AF.Pair bot B}› and
    all-heads-in-𝒩-are-bot: ‹(∀ 𝒞 ∈ 𝒩. F-of 𝒞 = bot)› and
    A-int-B: ‹A |∩| B = {||}› and
    A-neq: ‹A ≠ {||}›
    by auto
  have 𝒩-and-Pair-bot-A-entails-Pair-bot-B: ‹𝒩 ∪ {AF.Pair bot A} ⊨s_AF {AF.Pair
bot B}›
    using A-in-N-entails-B by auto
  let ?𝒞 = ‹AF.Pair (F-of 𝒞') B›
  have neg-entails-version:
    ‹core.enabled ?𝒞 J ⟹
        core.fml-ext ‘ total-strip J ∪ Pos ‘ (insert (AF.Pair (F-of 𝒞') A) 𝒩 proj_J
J)
          ⊨s_∼ {Pos (F-of ?𝒞)}›
    for J
  proof −
    fix J
    assume ‹core.enabled ?𝒞 J›
    then have B-in-J: ‹fset B ⊆ total-strip J›
      by (simp add: core.enabled-def)
      then consider (fml-unsat) ‹core.sound-cons.entails-neg (core.fml-ext ‘ to-
tal-strip J) {Pos bot}› |
      (𝒩-unsat) ‹∃ A' ∈ A-of ‘ 𝒩. fset A' ⊆ total-strip J› |
      (A-subset-J) ‹fset A ⊆ total-strip J›
       using core.strong-entails-bot-cases[OF 𝒩-and-Pair-bot-A-entails-Pair-bot-B,
rule-format,
          OF B-in-J]
        core.strong-entails-bot-cases-Union[rule-format, OF - - B-in-J]
    by (smt (verit, ccfv-SIG) Un-commute core.enabled-def core.enabled-projection-def
equals0I
        image-iff mem-Collect-eq sup-bot-left)
    then show
     ‹core.fml-ext ‘ total-strip J ∪ Pos ‘ (insert (AF.Pair (F-of 𝒞') A) 𝒩 proj_J J)
          ⊨s_∼ {Pos (F-of ?𝒞)}›
    proof cases
      case fml-unsat
      then have ‹(core.fml-ext ‘ total-strip J) ⊨s_∼ {Pos bot, Pos (F-of 𝒞')}›
        by (smt (verit, best) Un-absorb consequence-relation.entails-subsets in-
sert-is-Un
          core.sound-cons.ext-cons-rel sup-ge1)
      moreover have ‹((core.fml-ext ‘ total-strip J) ∪ {Pos bot}) ⊨s_∼ {Pos (F-of
𝒞')}›
       by (smt (verit, del-insts) Un-commute Un-upper2 empty-subsetI insert-subset
        mem-Collect-eq core.sound-cons.bot-entails-empty core.sound-cons.entails-neg-def
          core.sound-cons.entails-subsets)
      ultimately have ‹(core.fml-ext ‘ total-strip J) ⊨s_∼ {Pos (F-of 𝒞')}›
        using consequence-relation.entails-cut core.sound-cons.ext-cons-rel
```

```
      by fastforce
    then show ?thesis
      by (smt (verit, best) AF.sel(1) consequence-relation.entails-subsets
          insert-is-Un core.sound-cons.ext-cons-rel sup-ge1)
  next
    case 𝒩-unsat
    then have ‹bot ∈ 𝒩 proj_J J›
      unfolding core.enabled-projection-def core.enabled-def
      using all-heads-in-𝒩-are-bot by fastforce
    then have ‹(Pos ‘ (𝒩 proj_J J)) ⊨s_∼ {}›
      by (smt (verit, del-insts) consequence-relation.bot-entails-empty
       consequence-relation.entails-subsets image-insert insert-is-Un mk-disjoint-insert
          core.sound-cons.ext-cons-rel sup-bot-right sup-ge1)
    then show ?thesis
     by (smt (verit, ccfv-threshold) Un-upper2 consequence-relation.entails-subsets
            core.distrib-proj image-Un insert-is-Un core.sound-cons.ext-cons-rel
sup-assoc)
  next
    case A-subset-J
    then have pair-bot-A-enabled: ‹core.enabled (AF.Pair bot A) J›
      by (simp add: core.enabled-def)
    then have ‹{Pos (F-of 𝒞')} ⊨s_∼ {Pos (F-of 𝒞')}›
     by (meson consequence-relation.entails-reflexive core.sound-cons.ext-cons-rel)
    then have ‹(Pos ‘ ({AF.Pair (F-of 𝒞') A} proj_J J)) ⊨s_∼ {Pos (F-of 𝒞')}›
      using pair-bot-A-enabled  core.enabled-def core.enabled-projection-def
      by force
    then show ?thesis
      by (smt (verit, ccfv-threshold) AF.sel(1) Un-commute Un-upper2
       consequence-relation.entails-subsets core.distrib-proj image-Un insert-is-Un
        core.sound-cons.ext-cons-rel)
  qed
 qed
 have trim-assms: ‹A-of 𝒞' = A |∪| B ∧ F-of 𝒞' ≠ bot ∧ 𝒩 ∪ {AF.Pair bot A}
⊨s_AF {AF.Pair bot B} ∧
  (∀𝒞∈𝒩. F-of 𝒞 = bot) ∧ A |∩| B = {||} ∧ A ≠ {||}›
  using A-of-Cp F-of-Cp 𝒩-and-Pair-bot-A-entails-Pair-bot-B all-heads-in-𝒩-are-bot
A-int-B A-neq
   by blast
 have ‹𝒞' ▷ 𝒩 ⊨s_AF {?𝒞}›
   unfolding core.AF-entails-sound-def core.enabled-set-def
   by (smt (verit, ccfv-threshold) AF.collapse AF.sel(2) A-of-Cp core.distrib-proj
core.enabled-def
       core.projection-of-enabled-subset image-empty image-insert insertCI in-
sert-is-Un
      neg-entails-version)
 then show ‹insert 𝒞' 𝒩 ⊨s_AF {𝒞}›
   using C-is by auto
qed
```

theorem *trim-redundant*: ‹*A-of* $\mathcal{C}$ = *A* |∪| *B* ∧
F-of $\mathcal{C}$ ≠ *bot* ∧
$\mathcal{M}$ ∪ {*AF.Pair bot A*} ⊨s_{AF} {*AF.Pair bot B*} ∧
(∀$\mathcal{C}$∈$\mathcal{M}$. *F-of* $\mathcal{C}$ = *bot*) ∧ *A* |∩| *B* = {||} ∧ *A* ≠ {||} ⟹
$\mathcal{C}$ ∈ *core.SRed*$_F$ ($\mathcal{M}$ ∪ { *AF.Pair* (*F-of* $\mathcal{C}$) *B* })›
proof −
assume ‹*A-of* $\mathcal{C}$ = *A* |∪| *B* ∧ *F-of* $\mathcal{C}$ ≠ *bot* ∧ $\mathcal{M}$ ∪ {*AF.Pair bot A*} ⊨s_{AF} {*AF.Pair bot B*} ∧
(∀$\mathcal{C}$∈$\mathcal{M}$. *F-of* $\mathcal{C}$ = *bot*) ∧ *A* |∩| *B* = {||} ∧ *A* ≠ {||}›
then have *A-of-C-is*: ‹*A-of* $\mathcal{C}$ = *A* |∪| *B*› **and**
‹*F-of* $\mathcal{C}$ ≠ *bot*› **and**
$\mathcal{M}$*-and-A-entail-bot-B*: ‹*AF.Pair bot A* ▷ $\mathcal{M}$ ⊨s_{AF} {*AF.Pair bot B*}›
and
‹∀$\mathcal{C}$ ∈ $\mathcal{M}$. *F-of* $\mathcal{C}$ = *bot*› **and**
A-B-disjoint: ‹*A* |∩| *B* = {||}› **and**
A-not-empty: ‹*A* ≠ {||}›
by *auto*
then have ‹∃ $\mathcal{C}'$ ∈ $\mathcal{M}$ ∪ {*AF.Pair* (*F-of* $\mathcal{C}$) *B*}. (*F-of* $\mathcal{C}'$ = *F-of* $\mathcal{C}$ ∧ *A-of* $\mathcal{C}'$ |⊂| *A-of* $\mathcal{C}$)›
by *auto*
then show ‹$\mathcal{C}$ ∈ *core.SRed*$_F$ ($\mathcal{M}$ ∪ { *AF.Pair* (*F-of* $\mathcal{C}$) *B* })›
unfolding *core.SRed*$_F$*-def*
by (*smt* (*verit*, *del-insts*) *AF.collapse CollectI Un-iff insert-iff singletonD*)
qed

lemma *extend-simps-with-trim*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* (⊨$_{AF}$) (⊨s_{AF}) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-trim* (*to-AF bot*) *core.SInf* (⊨$_{AF}$) (⊨s_{AF}) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*›
proof −
interpret *sound-simps*:
AF-calculus-extended to-AF bot core.SInf (⊨$_{AF}$) (⊨s_{AF}) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*
using *assms* **.**
show *?thesis*
proof
fix $\mathcal{C}$ *A B* $\mathcal{M}$
assume ‹*A-of* $\mathcal{C}$ = *A* |∪| *B* ∧
F-of $\mathcal{C}$ ≠ *F-of* (*to-AF bot*) ∧
$\mathcal{M}$ ∪ {*AF.Pair* (*F-of* (*to-AF bot*)) *A*} ⊨s_{AF} {*AF.Pair* (*F-of* (*to-AF bot*)) *B*} ∧
(∀$\mathcal{C}$∈$\mathcal{M}$. *F-of* $\mathcal{C}$ = *F-of* (*to-AF bot*)) ∧ *A* |∩| *B* = {||} ∧ *A* ≠ {||}›
then show ‹$\mathcal{C}$ ▷ $\mathcal{M}$ ⊨s_{AF} {*AF.Pair* (*F-of* $\mathcal{C}$) *B*}›
using *trim-sound F-of-to-AF* **by** *metis*
next

fix $\mathcal{C}$ A B $\mathcal{M}$
assume ‹A-*of* $\mathcal{C}$ = A |∪| B ∧
F-*of* $\mathcal{C}$ ≠ F-*of* (*to-AF bot*) ∧
$\mathcal{M}$ ∪ {*AF.Pair* (F-*of* (*to-AF bot*)) A} $\models s_{AF}$ {*AF.Pair* (F-*of* (*to-AF bot*)) B} ∧
(∀$\mathcal{C}$∈$\mathcal{M}$. F-*of* $\mathcal{C}$ = F-*of* (*to-AF bot*)) ∧ A |∩| B = {||} ∧ A ≠ {||}›
then show ‹$\mathcal{C}$ ∈ *core.SRed*$_F$ ($\mathcal{M}$ ∪ {*AF.Pair* (F-*of* $\mathcal{C}$) B})›
using *trim-redundant F-of-to-AF* **by** *metis*
qed
qed

interpretation *splitting-calc-with-trim*:
AF-calculus-with-trim to-AF bot core.SInf ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ {} {}
using *extend-simps-with-trim*[*OF empty-simps.AF-calculus-extended-axioms*] **.**

end

9.2 Extra Inferences

We extend our basic splitting calculus with new optional rules:

- StrongUnsat is a variant of Unsat which uses the sound entailment instead of the "normal" entailment;
- Approx is a very special case for Split where $n = 1$;
- Tauto inserts a new formula which is always true.

9.2.1 The StrongUnsat Rule

locale *AF-calculus-with-strong-unsat* =
base: *AF-calculus-extended bot SInf entails entails-sound Red-I Red-F Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› **and**
entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*)
and
Red-I :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
Red-F :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**
Simps :: ‹('*f*, '*v*) *AF simplification set*› **and**
OptInfs :: ‹('*f*, '*v*) *AF inference set*›
begin

We follow the same naming conventions for our new inference rules as for the two inference rules defined in *annotated-calculus*. *X_inf* defines the inference rule, while *X_pre* is the precondition for the application of the inference rule.

abbreviation *strong-unsat-pre* :: ‹('f, 'v) *AF list* ⇒ *bool*› **where**
‹*strong-unsat-pre* $\mathcal{M}$ ≡ (*set* $\mathcal{M}$ $\models s_{AF}$ {*bot*}) ∧ (∀ *x* ∈ *set* $\mathcal{M}$. *F-of x* = (*F-of bot*))›

abbreviation *strong-unsat-inf* :: ‹('f, 'v) *AF list* ⇒ ('f, 'v) *AF inference*› **where**
‹*strong-unsat-inf* $\mathcal{M}$ ≡ *Infer* $\mathcal{M}$ *bot*›

Instead of considering an inference system with only the new rule, here StrongUnsat, we instead add it to the inference system provided so that it is possible to extend the system rule by rule in a modular way.

inductive-set *OptInfs-with-strong-unsat* :: ‹('f, 'v) *AF inference set*› **where**
strong-unsat: ‹*strong-unsat-pre* $\mathcal{M}$ ⟹ *strong-unsat-inf* $\mathcal{M}$ ∈ *OptInfs-with-strong-unsat*›
| *from-OptInf*: ‹*ι* ∈ *OptInfs* ⟹ *ι* ∈ *OptInfs-with-strong-unsat*›

theorem *OptInf-with-strong-unsat-sound-wrt-entails-sound*:
‹*ι* ∈ *OptInfs-with-strong-unsat* ⟹ *set* (*prems-of ι*) $\models s_{AF}$ {*concl-of ι*}›
proof −
assume ‹*ι* ∈ *OptInfs-with-strong-unsat*›
then show *?thesis*
proof (*cases ι rule*: *OptInfs-with-strong-unsat.cases*)
case (*strong-unsat* $\mathcal{M}$)
then show *?thesis*
by *simp*
next
case *from-OptInf*
then show *?thesis*
using *base.infs-sound* **by** *blast*
qed
qed

interpretation *AF-sound-cons-rel*: *consequence-relation bot* ‹($\models s_{AF}$)›
using *base.sc.sound-cons.consequence-relation-axioms* **.**

interpretation *SInf-sound-inf-system*: *sound-inference-system SInf bot* ‹($\models s_{AF}$)›
using *base.sc.sound-inference-system-axioms* **.**

definition *Red-I-ext* **where**
‹*Red-I-ext* $\mathcal{N}$ = *Red-I* $\mathcal{N}$ ∪ { *strong-unsat-inf* $\mathcal{M}$ | $\mathcal{M}$. *strong-unsat-pre* $\mathcal{M}$ ∧ *bot* ∈ $\mathcal{N}$ }›

interpretation *AF-calc-with-strong-unsat*:
AF-calculus bot SInf entails entails-sound Red-I Red-F
using *base.AF-calculus-axioms* **.**

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails entails-sound Red-I Red-F*
Simps OptInfs-with-strong-unsat
using *base.simps-simp base.simps-sound*

OptInf-with-strong-unsat-sound-wrt-entails-sound
by (*unfold-locales*, *auto*)

end

context *splitting-calculus*
begin

lemma *extend-infs-with-strong-unsat*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-strong-unsat* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps*
OptInfs›
using *AF-calculus-with-strong-unsat.intro assms* **by** *blast*

interpretation *splitting-calc-with-strong-unsat*: *AF-calculus-with-strong-unsat to-AF bot core.SInf*
($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ {} {}
using
extend-infs-with-strong-unsat[*OF empty-simps.AF-calculus-extended-axioms*] **.**

end

9.2.2 The Tauto Rule

locale *AF-calculus-with-tauto* =
base: *AF-calculus-extended bot SInf entails entails-sound Red-I Red-F Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› **and**
entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*)
and
Red-I :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
Red-F :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**
Simps :: ‹('*f*, '*v*) *AF simplification set*› **and**
OptInfs :: ‹('*f*, '*v*) *AF inference set*›
begin

abbreviation *tauto-pre* :: ‹('*f*, '*v*) *AF* ⇒ *bool*› **where**
‹*tauto-pre* $\mathcal{C}$ ≡ {} $\models s_{AF}$ { $\mathcal{C}$ }›

abbreviation *tauto-inf* :: ‹('*f*, '*v*) *AF* ⇒ ('*f*, '*v*) *AF inference*› **where**
‹*tauto-inf* $\mathcal{C}$ ≡ *Infer* [] $\mathcal{C}$›

inductive-set *OptInfs-with-tauto* :: ‹('f, 'v) *AF inference set*› **where**
  tauto: ‹*tauto-pre* $\mathcal{C}$ $\Longrightarrow$ *tauto-inf* $\mathcal{C}$ $\in$ *OptInfs-with-tauto*›
| *from-OptInfs*: ‹ι $\in$ *OptInfs* $\Longrightarrow$ ι $\in$ *OptInfs-with-tauto*›

theorem *OptInfs-with-tauto-sound-wrt-entails-sound*: ‹ι $\in$ *OptInfs-with-tauto* $\Longrightarrow$

  set (*prems-of* ι) $\models s_{AF}$ {*concl-of* ι}›
proof −
  assume ‹ι $\in$ *OptInfs-with-tauto*›
  then show *?thesis*
  proof (*cases* ι *rule*: *OptInfs-with-tauto.cases*)
    case (*tauto* $\mathcal{C}$)
    then show *?thesis*
      by *auto*
  next
    case *from-OptInfs*
    then show *?thesis*
      using *base.infs-sound* **by** *blast*
  qed
qed

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails* ($\models s_{AF}$) *Red-I*
  Red-F Simps OptInfs-with-tauto
 using *OptInfs-with-tauto-sound-wrt-entails-sound base.simps-sound base.simps-simp*

  by (*unfold-locales*, *auto*)

end

context *splitting-calculus*
begin

lemma *extend-infs-with-tauto*:
  assumes
    ‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
      core.SRed$_F$ *Simps OptInfs*›
  shows
   ‹*AF-calculus-with-tauto* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$
Simps OptInfs›
  using *AF-calculus-with-tauto.intro assms* **by** *blast*

interpretation *splitting-calc-with-tauto*:
  AF-calculus-with-tauto to-AF bot core.SInf ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$
{} {}
  using *extend-infs-with-tauto*[*OF empty-simps.AF-calculus-extended-axioms*] **.**

end

9.2.3 The Approx Rule

locale *AF-calculus-with-approx* =
  base: *AF-calculus-extended bot SInf entails entails-sound Red-I Red-F Simps OptInfs*
  for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
    SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
    entails :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› **and**
    entails-sound :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set* ⇒ *bool*› (**infix** ‹$\models s_{AF}$› *50*)
and
    Red-I :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF inference set*› **and**
    Red-F :: ‹('*f*, '*v*) *AF set* ⇒ ('*f*, '*v*) *AF set*› **and**
    Simps :: ‹('*f*, '*v*) *AF simplification set*› **and**
    OptInfs :: ‹('*f*, '*v*) *AF inference set*›
  + **fixes**
    approximates :: ‹'*v sign* ⇒ ('*f*, '*v*) *AF* ⇒ *bool*›
  assumes
    approx-sound: ‹*approximates a* $\mathcal{C}$ ⟹ {$\mathcal{C}$} $\models s_{AF}$ {*AF.Pair* (*F-of bot*) (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))}›
begin

abbreviation *approx-pre* :: ‹'*v sign* ⇒ ('*f*, '*v*) *AF* ⇒ *bool*› **where**
  ‹*approx-pre a* $\mathcal{C}$ ≡ *approximates a* $\mathcal{C}$›

abbreviation *approx-inf* :: ‹('*f*, '*v*) *AF* ⇒ '*v sign* ⇒ ('*f*, '*v*) *AF inference*› **where**
  ‹*approx-inf* $\mathcal{C}$ *a* ≡ *Infer* [$\mathcal{C}$] (*AF.Pair* (*F-of bot*) (*finsert* (*neg a*) (*A-of* $\mathcal{C}$)))›

inductive-set *OptInfs-with-approx* :: ‹('*f*, '*v*) *AF inference set*› **where**
  approx: ‹*approx-pre a* $\mathcal{C}$ ⟹ *approx-inf* $\mathcal{C}$ *a* ∈ *OptInfs-with-approx*›
| *from-OptInfs*: ‹ι ∈ *OptInfs* ⟹ ι ∈ *OptInfs-with-approx*›

theorem *OptInfs-with-approx-sound-wrt-entails-sound*:
  ‹ι ∈ *OptInfs-with-approx* ⟹ *set* (*prems-of* ι) $\models s_{AF}$ {*concl-of* ι}›
proof −
  assume ‹ι ∈ *OptInfs-with-approx*›
  then show *?thesis*
  proof (*cases* ι *rule*: *OptInfs-with-approx.cases*)
    case (*approx a* $\mathcal{C}$)
    show *?thesis*
      using *approx-sound*[*OF approx*(*2*)] *approx*(*1*) **by** *simp*
  next
    case *from-OptInfs*
    show *?thesis*
      using *base.infs-sound*[*OF from-OptInfs*] **.**
  qed

qed

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails* ($\models s_{AF}$) *Red-I Red-F Simps*
OptInfs-with-approx
using *OptInfs-with-approx-sound-wrt-entails-sound base.simps-sound base.simps-simp*

by (*unfold-locales*, *auto*)

end

context *splitting-calculus*
begin

definition *approximates* :: ‹'*v sign* ⇒ ('*f*, '*v*) *AF* ⇒ *bool*› **where**
‹*approximates a* $\mathcal{C}$ ≡ *a* ∈ *asn* (*Pos* (*F-of* $\mathcal{C}$))›

lemma *approx-sound*: ‹*approximates a* $\mathcal{C}$ ⟹ {$\mathcal{C}$} $\models s_{AF}$ {*AF.Pair bot* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))}›
proof −
assume *approx-a-C*: ‹*approximates a* $\mathcal{C}$›
then have
‹*core.enabled* (*AF.Pair bot* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))) *J* ⟹
(*core.fml-ext* ‘ *total-strip J*) ∪ {*Pos* (*F-of* $\mathcal{C}$)} $\models s_{\sim}$ {*Pos bot*}›
for *J*
proof −
fix *J*
assume ‹*core.enabled* (*AF.Pair bot* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))) *J*›
then have ‹*fset* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$)) ⊆ *total-strip J*›
unfolding *core.enabled-def*
by *simp*
then have *neg-fml-ext-in-J*: ‹*neg* (*core.fml-ext a*) ∈ *core.fml-ext* ‘ *total-strip J*›
by (*smt* (*verit*, *ccfv-threshold*) *finsert.rep-eq core.fml-ext.elims core.fml-ext.simps*(*1*) *core.fml-ext.simps*(*2*) *image-iff insert-subset neg.simps*(*1*) *neg.simps*(*2*))
moreover have ‹{*Pos* (*F-of* $\mathcal{C}$)} $\models s_{\sim}$ {*Pos* (*F-of* $\mathcal{C}$)}›
using *core.equi-entails-if-a-in-asns approx-a-C* **unfolding** *approximates-def*
by *blast*
then have ‹(*core.fml-ext* ‘ (*total-strip J* − {*neg a*})) ∪ {*Pos* (*F-of* $\mathcal{C}$)} $\models s_{\sim}$ {*Pos bot*, *Pos* (*F-of* $\mathcal{C}$)}›
by (*metis* (*no-types*, *lifting*) *consequence-relation.entails-subsets insert-is-Un core.sound-cons.ext-cons-rel sup.cobounded2*)
ultimately show ‹(*core.fml-ext* ‘ *total-strip J*) ∪ {*Pos* (*F-of* $\mathcal{C}$)} $\models s_{\sim}$ {*Pos bot*}›
proof −
have ‹*core.fml-ext* ‘ *total-strip J* ∪ {*core.fml-ext a*} $\models s_{\sim}$ ({*Pos bot*} ∪ {})›
by (*smt* (*z3*) *Bex-def-raw UnCI Un-commute Un-insert-right Un-upper2 neg-fml-ext-in-J*
consequence-relation.entails-subsets insert-is-Un insert-subset
core.sound-cons.ext-cons-rel core.sound-cons.pos-neg-entails-bot)

then show *?thesis*
using *approx-a-C* **unfolding** *approximates-def*
by (*smt* (*verit*) *Un-commute Un-empty-right core.C-entails-fml core.fml-ext-is-mapping*

core.neg-ext-sound-cons-rel.entails-cut)
qed
qed
then have
‹*core.enabled-set* {*AF.Pair bot* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))} *J* $\Longrightarrow$
core.fml-ext ‘ *total-strip J* $\cup$ *Pos* ‘ ({$\mathcal{C}$} *proj*$_J$ *J*) $\models s_\sim$ {*Pos bot*}›
for *J*
unfolding *core.enabled-projection-def core.enabled-def core.enabled-set-def*
by *auto*
then show ‹{$\mathcal{C}$} $\models s_{AF}$ {*AF.Pair bot* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))}›
unfolding *core.AF-entails-sound-def* **by** *auto*
qed

lemma *extend-infs-with-approx*:
assumes
‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps*
OptInfs›
shows
‹*AF-calculus-with-approx* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps*
OptInfs approximates›
using *AF-calculus-with-approx.intro*[*OF assms*] *approx-sound*
by (*simp add*: *AF-calculus-with-approx-axioms-def F-of-to-AF*)

interpretation *splitting-calc-with-approx*:
AF-calculus-with-approx to-AF bot core.SInf ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ {} {}
approximates
using *extend-infs-with-approx*[*OF empty-simps.AF-calculus-extended-axioms*] **.**

end

9.3 Combining all simplifications and optional inferences

We have augmented the core calculus with each simplification and optional rule separately. We now show how to augment the core calculus with all of them in succession.

context *splitting-calculus*
begin

interpretation *with-A*: *AF-calculus-with-approx to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
core.SRed$_F$ {} {} *approximates*
using *extend-infs-with-approx*[*OF empty-simps.AF-calculus-extended-axioms*] **.**

interpretation *with-AT*: *AF-calculus-with-tauto to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
core.SRed$_F$ {} *with-A.OptInfs-with-approx*
using
extend-infs-with-tauto[*OF with-A.AF-calc-ext.AF-calculus-extended-axioms*] **.**

interpretation *with-ATS*: *AF-calculus-with-strong-unsat to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$)
core.SRed$_I$ *core.SRed*$_F$ {} *with-AT.OptInfs-with-tauto*
using *extend-infs-with-strong-unsat*[*OF*
with-AT.AF-calc-ext.AF-calculus-extended-axioms] **.**

interpretation *with-ATS-T*: *AF-calculus-with-trim to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
core.SRed$_F$ {} *with-ATS.OptInfs-with-strong-unsat*
using *extend-simps-with-trim*[*OF*
with-ATS.AF-calc-ext.AF-calculus-extended-axioms] **.**

interpretation *with-ATS-TC*: *AF-calculus-with-collect to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$)
core.SRed$_I$ *core.SRed*$_F$ *with-ATS-T.Simps-with-Trim with-ATS.OptInfs-with-strong-unsat*
using *extend-simps-with-collect*[*OF*
with-ATS-T.AF-calc-ext.AF-calculus-extended-axioms] **.**

interpretation *with-all*: *AF-calculus-with-split to-AF bot core.SInf* ($\models_{AF}$) ($\models s_{AF}$)

core.SRed$_I$ *core.SRed*$_F$ *with-ATS-TC.Simps-with-Collect with-ATS.OptInfs-with-strong-unsat splittable*
using *extend-simps-with-split*[*OF*
with-ATS-TC.AF-calc-ext.AF-calculus-extended-axioms] **.**

interpretation *full-splitting-calculus*: *AF-calculus-extended to-AF bot*
core.SInf ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *with-all.Simps-with-Split*
with-ATS.OptInfs-with-strong-unsat
using *with-all.AF-calc-ext.AF-calculus-extended-axioms* **.**

Simplifications and optional inferences can be integrated in derivations. This is made obvious by the following two lemmas.

lemma *simp-in-derivations*: ‹$\delta \in$ *with-all.Simps-with-Split* $\Longrightarrow$
core.S-calculus.derive ($\mathcal{M} \cup$ *S-from* δ) ($\mathcal{M} \cup$ *S-to* δ)›
unfolding *core.S-calculus.derive-def*
proof
fix *C*
assume *d-in*: ‹$\delta \in$ *with-all.Simps-with-Split*› **and**
‹$C \in \mathcal{M} \cup$ *S-from* $\delta -$ ($\mathcal{M} \cup$ *S-to* δ)›
then have ‹$C \in$ *S-from* $\delta -$ *S-to* δ›
by *blast*
then show ‹$C \in$ *core.SRed*$_F$ ($\mathcal{M} \cup$ *S-to* δ)›

using *with-all.AF-calc-ext.simps-simp*[*OF d-in*] **by** (*meson core.annotated-calculus-axioms annotated-calculus.SRed$_F$-of-subset-F in-mono inf-sup-ord*(*4*))
qed

lemma *opt-infs-in-derivations*: ‹$\iota \in$ *with-ATS.OptInfs-with-strong-unsat* $\Longrightarrow$
core.S-calculus.derive ($\mathcal{M} \cup$ *set* (*prems-of* ι)) ($\mathcal{M} \cup$ *set* (*prems-of* ι) $\cup$ {*concl-of* ι})›
unfolding *core.S-calculus.derive-def*
proof
fix C
assume ‹$\iota \in$ *with-ATS.OptInfs-with-strong-unsat*› **and**
C-in: ‹$C \in \mathcal{M} \cup$ *set* (*prems-of* ι) $-$ ($\mathcal{M} \cup$ *set* (*prems-of* ι) $\cup$ {*concl-of* ι})›
have ‹$\mathcal{M} \cup$ *set* (*prems-of* ι) $-$ ($\mathcal{M} \cup$ *set* (*prems-of* ι) $\cup$ {*concl-of* ι}) $=$ {}›
by *blast*
then have *False* **using** *C-in* **by** *auto*
then show ‹$C \in$ *core.SRed$_F$* ($\mathcal{M} \cup$ *set* (*prems-of* ι) $\cup$ {*concl-of* ι})›
by *auto*
qed

end

9.4 The BinSplit Simplification Rule

For the sake of the Lightweight Avatar calculus, we define the BinSplit simplification rule, and show that it is indeed a sound simplification rule as in the case of the Split rule.

locale *AF-calculus-with-binsplit* =
base-calculus: *AF-calculus-extended bot SInf entails entails-sound SRed$_I$ SRed$_F$ Simps OptInfs*
for *bot* :: ‹('*f*, '*v* :: *countable*) *AF*› **and**
SInf :: ‹('*f*, '*v*) *AF inference set*› **and**
entails :: ‹('*f*, '*v*) *AF set* $\Rightarrow$ ('*f*, '*v*) *AF set* $\Rightarrow$ *bool*› (**infix** ‹$\models_{AF}$› *50*) **and**
entails-sound :: ‹('*f*, '*v*) *AF set* $\Rightarrow$ ('*f*, '*v*) *AF set* $\Rightarrow$ *bool*› (**infix** ‹$\models s_{AF}$› *50*)
and
SRed$_I$ :: ‹('*f*, '*v*) *AF set* $\Rightarrow$ ('*f*, '*v*) *AF inference set*› **and**
SRed$_F$ :: ‹('*f*, '*v*) *AF set* $\Rightarrow$ ('*f*, '*v*) *AF set*› **and**
Simps :: ‹('*f*, '*v*) *AF simplification set*› **and**
OptInfs :: ‹('*f*, '*v*) *AF inference set*›
+ **fixes**
bin-splittable :: ‹('*f*, '*v*) *AF* $\Rightarrow$ '*f* $\Rightarrow$ '*f* $\Rightarrow$ ('*f*, '*v*) *AF fset* $\Rightarrow$ *bool*›
assumes
binsplit-prem-entails-cons: ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s \Longrightarrow \forall\ \mathcal{C}' \in$ *fset* $\mathcal{C}s$. {$\mathcal{C}$} $\models s_{AF}$ {$\mathcal{C}'$}› **and**
binsplit-simp: ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s \Longrightarrow \mathcal{C} \in$ *SRed$_F$* (*fset* $\mathcal{C}s$)›
begin

We use the same naming convention as for the other rules, where *X_pre* is the condition which must hold for the rule to be applicable, *X_res* is its resultant and *X_simp* is the simplification rule itself.

abbreviation *bin-split-pre* :: ‹('f, 'v) *AF* ⇒ 'f ⇒ 'f ⇒ ('f, 'v) *AF fset* ⇒ *bool*›
where
‹*bin-split-pre* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ≡ *bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$›

abbreviation *bin-split-res* :: ‹('f, 'v) *AF* ⇒ ('f, 'v) *AF fset* ⇒ ('f, 'v) *AF set*›
where
‹*bin-split-res* $\mathcal{C}$ $\mathcal{C}s$ ≡ *fset* $\mathcal{C}s$›

abbreviation *bin-split-simp* :: ‹('f, 'v) *AF* ⇒ ('f, 'v) *AF fset* ⇒ ('f, 'v) *AF simplification*› **where**
‹*bin-split-simp* $\mathcal{C}$ $\mathcal{C}s$ ≡ *Simplify* {$\mathcal{C}$} (*bin-split-res* $\mathcal{C}$ $\mathcal{C}s$)›

inductive-set *Simps-with-BinSplit* :: ‹('f, 'v) *AF simplification set*› **where**
binsplit: ‹*bin-split-pre* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ⟹ *bin-split-simp* $\mathcal{C}$ $\mathcal{C}s$ ∈ *Simps-with-BinSplit*›

| *other*: ‹*simp* ∈ *Simps* ⟹ *simp* ∈ *Simps-with-BinSplit*›

theorem *SInf-with-binsplit-sound-wrt-entails-sound*:
‹ι ∈ *Simps-with-BinSplit* ⟹ ∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι ⊨s_{AF} {$\mathcal{C}$}›
proof −
assume *ι-is-simp-rule*: ‹ι ∈ *Simps-with-BinSplit*›
then show ‹∀ $\mathcal{C}$ ∈ *S-to* ι. *S-from* ι ⊨s_{AF} {$\mathcal{C}$}›
proof (*intro ballI*)
fix $\mathcal{C}$
assume *C-is-consq-of-ι*: ‹$\mathcal{C}$ ∈ *S-to* ι›
show ‹*S-from* ι ⊨s_{AF} {$\mathcal{C}$}›
using *ι-is-simp-rule*
proof (*cases rule*: *Simps-with-BinSplit.cases*)
case (*binsplit* $\mathcal{C}'$ *C1 C2* $\mathcal{C}s$)
then have ‹∀$\mathcal{C}'$ ∈ *fset* $\mathcal{C}s$. *S-from* ι ⊨s_{AF} {$\mathcal{C}'$}›
using *binsplit-prem-entails-cons* **by** *auto*
then show *?thesis*
using *C-is-consq-of-ι binsplit* **unfolding** *S-to-def* **by** *auto*
next
case *other*
then show *?thesis*
using *C-is-consq-of-ι base-calculus.simps-sound* **by** *auto*
qed
qed
qed

lemma *binsplit-redundant*: ‹*bin-split-pre* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ⟹ $\mathcal{C}$ ∈ *SRed*$_F$ (*bin-split-res* $\mathcal{C}$ $\mathcal{C}s$)›
proof −
assume *pre-cond*: ‹*bin-split-pre* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$›

then show ‹$\mathcal{C} \in SRed_F$ (*bin-split-res* $\mathcal{C}$ $\mathcal{C}s$)›
using *binsplit-simp* **by** *simp*
qed

lemma *simps-with-binsplit-are-simps*:
‹$\iota \in$ *Simps-with-BinSplit* $\Longrightarrow$ (*S-from* ι − *S-to* ι) $\subseteq SRed_F$ (*S-to* ι)›
proof
fix $\mathcal{C}$
assume *i-in*: ‹$\iota \in$ *Simps-with-BinSplit*› **and**
C-in: ‹$\mathcal{C} \in$ *S-from* ι − *S-to* ι›
then show ‹$\mathcal{C} \in SRed_F$ (*S-to* ι)›
proof (*cases rule*: *Simps-with-BinSplit.cases*)
case (*binsplit* $\mathcal{C}'$ *C1 C2* $\mathcal{C}s$)
then have ‹$\mathcal{C} = \mathcal{C}'$› **using** *C-in* **by** *auto*
moreover have ‹*S-to* ι = *bin-split-res* $\mathcal{C}'$ $\mathcal{C}s$› **using** *binsplit*(*1*) *simplification.sel*(*2*) **by** *auto*
ultimately show *?thesis*
using *binsplit-redundant*[*OF binsplit*(*2*)] **by** *presburger*
next
case *other*
then show *?thesis*
using *base-calculus.simps-simp C-in* **by** *blast*
qed
qed

sublocale *AF-calc-ext*: *AF-calculus-extended bot SInf entails entails-sound* $SRed_I$ $SRed_F$ *Simps-with-BinSplit OptInfs*
using *simps-with-binsplit-are-simps SInf-with-binsplit-sound-wrt-entails-sound base-calculus.infs-sound* **by** (*unfold-locales*, *auto*)

end

context *splitting-calculus*
begin

definition *mk-bin-split* :: ‹('*f*, '*v*) *AF* ⇒ '*f* ⇒ '*f* ⇒ ('*f*, '*v*) *AF fset*› **where**
‹*split-form* (*F-of* $\mathcal{C}$) {|*C1*, *C2*|} $\Longrightarrow$ *mk-bin-split* $\mathcal{C}$ *C1 C2* ≡
let a = (*SOME a*. *a* ∈ *asn* (*sign.Pos C1*))
in {|*AF.Pair C1* (*finsert a* (*A-of* $\mathcal{C}$)), *AF.Pair C2* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))|}›

definition *bin-splittable* :: ‹('*f*, '*v*) *AF* ⇒ '*f* ⇒ '*f* ⇒ ('*f*, '*v*) *AF fset* ⇒ *bool*› **where**
‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ ≡ *split-form* (*F-of* $\mathcal{C}$) {|*C1*, *C2*|} ∧ *mk-bin-split* $\mathcal{C}$ *C1 C2* = $\mathcal{C}s$›

theorem *binsplit-prem-entails-cons*: ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$ $\Longrightarrow$ ∀ $\mathcal{C}'$ ∈ *fset* $\mathcal{C}s$. {$\mathcal{C}$} $\models s_{AF}$ {$\mathcal{C}'$}›

proof (*intro ballI*)
fix $\mathcal{C}'$
assume *C-u-D-splittable*: ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s$› **and**
C′-in: ‹$\mathcal{C}'$ |∈| $\mathcal{C}s$›
then have *split-fm*: ‹*split-form* (*F-of* $\mathcal{C}$) {|*C1*, *C2*|}› **and**
make-split: ‹*mk-bin-split* $\mathcal{C}$ *C1 C2* = $\mathcal{C}s$›
unfolding *bin-splittable-def* **by** *blast+*
have ‹$\mathcal{C}'$ ∈ (*let a* = *SOME a*. *a* ∈ *asn* (*sign.Pos C1*)
in {*AF.Pair C1* (*finsert a* (*A-of* $\mathcal{C}$)), *AF.Pair C2* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))})›
using *make-split mk-bin-split-def*[*OF split-fm*] *C′-in* **by** (*metis bot-fset.rep-eq fset-simps*(*2*))
then obtain *a* **where**
a-in-asn-pos-C1: ‹*a* ∈ *asn* (*sign.Pos C1*)› **and**
C′-is: ‹$\mathcal{C}'$ = *AF.Pair C1* (*finsert a* (*A-of* $\mathcal{C}$)) ∨ $\mathcal{C}'$ = *AF.Pair C2* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))›
using *core.asn-not-empty insert-iff singletonD some-in-eq* **by** *metis*
consider (*C1*) ‹$\mathcal{C}'$ = *AF.Pair C1* (*finsert a* (*A-of* $\mathcal{C}$))›
| (*C2*) ‹$\mathcal{C}'$ = *AF.Pair C2* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))›
using *C′-is* **by** *blast*
then show ‹{$\mathcal{C}$} $\models s_{AF}$ {$\mathcal{C}'$}›
unfolding *core.AF-entails-sound-def core.enabled-set-def core.enabled-def*
proof (*cases*)
case *C1*
assume *Cp-eq*: ‹$\mathcal{C}'$ = *AF.Pair C1* (*finsert a* (*A-of* $\mathcal{C}$))›
show ‹∀ *J*. (∀$\mathcal{C}$∈{$\mathcal{C}'$}. *fset* (*A-of* $\mathcal{C}$) ⊆ *total-strip J*) ⟶
core.fml-ext ‘ *total-strip J* ∪ *Pos* ‘ ({$\mathcal{C}$} *proj*$_J$ *J*) $\models s_\sim$ *Pos* ‘ *F-of* ‘ {$\mathcal{C}'$}›
proof (*rule allI*, *rule impI*)
fix *J*
assume ‹∀$\mathcal{C}$∈{$\mathcal{C}'$}. *fset* (*A-of* $\mathcal{C}$) ⊆ *total-strip J*›
then have ‹*a* ∈ *total-strip J*›
and ‹*fset* (*A-of* $\mathcal{C}$) ⊆ *total-strip J*›
using *Cp-eq* **by** *auto*
then have ‹*core.fml-ext* ‘ *total-strip J* ∪ *sign.Pos* ‘ ({$\mathcal{C}$} *proj*$_J$ *J*) $\models s_\sim$ {*sign.Pos C1*}›
using *a-in-asn-pos-C1* **by** (*smt* (*verit*, *best*) *core.fml-entails-C core.fml-ext-is-mapping core.neg-ext-sound-cons-rel.entails-subsets image-eqI singletonD subsetI sup-ge1*)
then show ‹*core.fml-ext* ‘ *total-strip J* ∪ *Pos* ‘ ({$\mathcal{C}$} *proj*$_J$ *J*) $\models s_\sim$ *Pos* ‘ *F-of* ‘ {$\mathcal{C}'$}›
by (*simp add*: *Cp-eq*)
qed
next
case *C2*
assume *C′-is-C2*: ‹$\mathcal{C}'$ = *AF.Pair C2* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))›
show ‹∀ *J*. (∀$\mathcal{C}$∈{$\mathcal{C}'$}. *fset* (*A-of* $\mathcal{C}$) ⊆ *total-strip J*) ⟶
core.fml-ext ‘ *total-strip J* ∪ *sign.Pos* ‘ ({$\mathcal{C}$} *proj*$_J$ *J*) $\models s_\sim$ *sign.Pos* ‘ *F-of* ‘ {$\mathcal{C}'$}›

```
    proof (rule allI, rule impI)
      fix J
      assume ‹∀C∈{C'}. fset (A-of C) ⊆ total-strip J›
      then have a-notin: ‹a ∉ total-strip J›
        and in-J: ‹fset (A-of C) ⊆ total-strip J›
        using C'-is-C2 by auto
      have ‹{F-of C} ⊨s fset {|C1, C2|}›
        using split-fm unfolding split-form-def by blast
      then have ‹{sign.Neg C1} ∪ {sign.Pos (F-of C)} ⊨s∼ {sign.Pos C2}›
        unfolding core.sound-cons.entails-neg-def by simp
     moreover have neg-fml-a-in: ‹neg (map-sign fml a) ∈ core.fml-ext ‘ total-strip
J›
      using a-notin by (smt (z3) core.fml-ext.elims core.fml-ext.simps(1) core.fml-ext.simps(2)

        core.fml-ext-is-mapping image-iff neg.simps(1) neg.simps(2) neg-in-total-strip)
      ultimately have neg-fml-a-in-entails:
        ‹{neg (map-sign fml a)} ∪ {sign.Pos (F-of C)} ⊨s∼ {sign.Pos C2}›
        using core.fml-entails-C core.C-entails-fml
        by (metis a-in-asn-pos-C1 consequence-relation.swap-neg-in-entails-neg
         core.neg-ext-sound-cons-rel.entails-cut core.sound-cons.consequence-relation-axioms

            insert-is-Un neg.simps(1) sup-commute)
      have ‹core.fml-ext ‘ total-strip J ∪ {sign.Pos (F-of C)} ⊨s∼ {sign.Pos C2}›
      proof −
        have f1: ∀ S. neg (map-sign fml a) ▷ S ⊆ S ∪ core.fml-ext ‘ total-strip J
          using neg-fml-a-in by blast
        have ∀ S s Sa. (s::'f sign) ▷ Sa ⊆ Sa ∪ (s ▷ S)
          by force
        then show ?thesis
          using f1
          by (metis (no-types) neg-fml-a-in-entails
              core.neg-ext-sound-cons-rel.entails-subsets insert-is-Un sup-ge1)
      qed
      moreover have ‹{C} proj_J J = {(F-of C)}›
        using in-J unfolding core.enabled-projection-def core.enabled-def by blast
      ultimately have ‹core.fml-ext ‘ total-strip J ∪ sign.Pos ‘ ({C} proj_J J) ⊨s∼
{sign.Pos C2}›
        by simp
      then show ‹core.fml-ext ‘ total-strip J ∪ Pos ‘ ({C} proj_J J) ⊨s∼ Pos ‘ F-of
‘ {C'}›
        using C'-is-C2 by auto
    qed
  qed
qed

theorem binsplit-simp: ‹bin-splittable C C1 C2 Cs ⟹ C ∈ core.SRed_F (fset Cs)›
proof −
  assume ‹bin-splittable C C1 C2 Cs›
  then have split-fm: ‹split-form (F-of C) {|C1, C2|}› and
```

make-split: ‹*mk-bin-split* $\mathcal{C}$ *C1 C2* = $\mathcal{C}s$›
unfolding *bin-splittable-def* **by** *blast+*
have *F-entailment*: ‹{*F-of* $\mathcal{C}$} $\models s$ *fset* {|*C1*, *C2*|}›
using *split-fm* **unfolding** *split-form-def* **by** *blast*
have *C-D-make-C-u-D-redundant*: ‹$\forall$ *C'*. *C'* |$\in$| {|*C1*, *C2*|} $\longrightarrow$ *F-of* $\mathcal{C}$ $\in$ Red_F {*C'*}›
by (*meson split-fm split-form-def*)
have *a-ex*: ‹$\exists$ *a* $\in$ *asn* (*sign.Pos C1*).
fset $\mathcal{C}s$ = {*AF.Pair C1* (*finsert a* (*A-of* $\mathcal{C}$)), *AF.Pair C2* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))}›
by (*metis mk-bin-split-def*[*OF split-fm*] *bot-fset.rep-eq core.asn-not-empty finsert.rep-eq*
make-split some-in-eq)
then obtain *a* **where** *a-in*: ‹*a* $\in$ *asn* (*sign.Pos C1*)› **and**
$\mathcal{C}s$*-is*: ‹*fset* $\mathcal{C}s$ = {*AF.Pair C1* (*finsert a* (*A-of* $\mathcal{C}$)), *AF.Pair C2* (*finsert* (*neg a*) (*A-of* $\mathcal{C}$))}›
by *blast*
show ‹$\mathcal{C}$ $\in$ *core.SRed*$_F$ (*fset* $\mathcal{C}s$)›
proof −
have ‹$\forall \mathcal{J}$. *fset* (*A-of* $\mathcal{C}$) $\subseteq$ *total-strip* $\mathcal{J}$ $\longrightarrow$ (*F-of* $\mathcal{C}$) $\in$ Red_F ((*fset* $\mathcal{C}s$) $proj_J$ $\mathcal{J}$)›
proof (*intro allI impI*)
fix $\mathcal{J}$
assume *A-in-*$\mathcal{J}$: ‹*fset* (*A-of* $\mathcal{C}$) $\subseteq$ *total-strip* $\mathcal{J}$›
then have *a-or-neg-a-in-*$\mathcal{J}$: ‹*a* $\in$ *total-strip* $\mathcal{J}$ $\vee$ *neg a* $\in$ *total-strip* $\mathcal{J}$›
by *simp*
then have *a-or-neg-a-in-*$\mathcal{J}$: ‹*fset* (*finsert a* (*A-of* $\mathcal{C}$)) $\subseteq$ *total-strip* $\mathcal{J}$ $\vee$
fset (*finsert* (*neg a*) (*A-of* $\mathcal{C}$)) $\subseteq$ *total-strip* $\mathcal{J}$›
by (*simp add*: *A-in-*$\mathcal{J}$)
have ‹(*fset* $\mathcal{C}s$) $proj_J$ $\mathcal{J}$ $\subseteq$ {*C1*, *C2*}›
using $\mathcal{C}s$*-is core.enabled-projection-def*
by *auto*
moreover have ‹(*fset* $\mathcal{C}s$) $proj_J$ $\mathcal{J}$ $\neq$ {}›
unfolding *core.enabled-projection-def* **using** *a-or-neg-a-in-*$\mathcal{J}$ $\mathcal{C}s$*-is*
by (*metis* (*mono-tags*, *lifting*) *AF.sel*(*2*) *core.enabled-def empty-iff insertCI mem-Collect-eq*)
ultimately show ‹(*F-of* $\mathcal{C}$) $\in$ Red_F ((*fset* $\mathcal{C}s$) $proj_J$ $\mathcal{J}$)›
using *C-D-make-C-u-D-redundant*
by (*smt* (*verit*, *del-insts*) *core.Red-F-of-subset all-not-in-conv empty-subsetI finsertCI insert-iff insert-subset subsetD*)
qed
then show *?thesis*
unfolding *core.SRed*$_F$*-def* **by** (*smt* (*verit*, *ccfv-threshold*) *AF.collapse UnCI mem-Collect-eq*)
qed
qed

lemma *extend-simps-with-binsplit*:
assumes

‹*AF-calculus-extended* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
core.SRed$_F$ *Simps OptInfs*›
shows
‹*AF-calculus-with-binsplit* (*to-AF bot*) *core.SInf* ($\models_{AF}$) ($\models s_{AF}$) *core.SRed*$_I$
core.SRed$_F$ *Simps*
OptInfs bin-splittable›
proof −
interpret *sound-simps*: *AF-calculus-extended to-AF bot core.SInf* ($\models_{AF}$)
($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ *Simps OptInfs*
using *assms* **.**
show *?thesis*
proof
show ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s \Longrightarrow \forall \mathcal{C}' \in |\mathcal{C}s.\ \{\mathcal{C}\} \models s_{AF}\ \{\mathcal{C}'\}$› **for** $\mathcal{C}$ *C1 C2*
$\mathcal{C}s$
using *binsplit-prem-entails-cons* **.**
next
show ‹*bin-splittable* $\mathcal{C}$ *C1 C2* $\mathcal{C}s \Longrightarrow \mathcal{C} \in$ *core.SRed*$_F$ (*fset* $\mathcal{C}s$)› **for** $\mathcal{C}$ *C1 C2*
$\mathcal{C}s$
using *binsplit-simp* **.**
qed
qed

interpretation *splitting-calc-with-binsplit*: *AF-calculus-with-binsplit to-AF bot core.SInf*
($\models_{AF}$)
($\models s_{AF}$) *core.SRed*$_I$ *core.SRed*$_F$ {} {} *bin-splittable*
using *extend-simps-with-binsplit*[*OF empty-simps.AF-calculus-extended-axioms*]
.

end

end

theory *Lightweight-Avatar*
imports
Main
Modular-Splitting-Calculus
Saturation-Framework-Extensions.FO-Ordered-Resolution-Prover-Revisited
begin

9.5 Ordered Resolution with a Disjunctive Consequence Relation

locale *FO-resolution-prover-disjunctive* = *FO-resolution-prover S subst-atm id-subst*
comp-subst
renaming-aparts atm-of-atms mgu less-atm
for
S :: ‹('*a* :: *wellorder*) *clause* ⇒ '*a clause*› **and**

subst-atm :: ‹'*a* ⇒ '*s* ⇒ '*a*› **and**
id-subst :: ‹'*s*› **and**
comp-subst :: ‹'*s* ⇒ '*s* ⇒ '*s*› **and**
renaming-aparts :: ‹'*a clause list* ⇒ '*s list*› **and**
atm-of-atms :: ‹'*a list* ⇒ '*a*› **and**
mgu :: ‹'*a set set* ⇒ '*s option*› **and**
less-atm :: ‹'*a* ⇒ '*a* ⇒ *bool*›
begin

no-notation *entails-clss* (**infix** ‹⊫*e*› *50*)
no-notation *Sema.entailment* (‹(- ⊫/ -)› [*53*, *53*] *53*)
no-notation *Linear-Temporal-Logic-on-Streams.HLD-nxt* (**infixr** · *65*)

notation *entails-clss* (**infix** ‹⊫∩*e*› *50*)

interpretation *gr*: *ground-resolution-with-selection S-M S M*
using *selection-axioms* **by** *unfold-locales* (*fact S-M-selects-subseteq S-M-selects-neg-lits*)+

interpretation *G*: *Soundness.sound-inference-system G-Inf M* {{#}} (⊫∩*e*)
proof
fix *ι*
assume *i-in*: *ι* ∈ *G-Inf M*
moreover
{
fix *I*
assume *I-ent-prems*: *I* ⊫*s set* (*prems-of ι*)
obtain *CAs AAs As* **where**
the-inf: *gr.ord-resolve M CAs* (*main-prem-of ι*) *AAs As* (*concl-of ι*) **and**
CAs: *CAs* = *side-prems-of ι*
using *i-in* **unfolding** *G-Inf-def* **by** *auto*
then have *I* ⊫ *concl-of ι*
using *gr.ord-resolve-sound*[*of M CAs main-prem-of ι AAs As concl-of ι I*]
by (*metis I-ent-prems G-Inf-have-prems i-in insert-is-Un set-mset-mset set-prems-of*
true-clss-insert true-clss-set-mset)
}
ultimately show *set* (*inference.prems-of ι*) ⊫∩*e* {*concl-of ι*}
by *simp*
qed

interpretation *G*: *clausal-counterex-reducing-inference-system G-Inf M gr.INTERP M*
proof
fix *N D*
assume
{#} ∉ *N* **and**
D ∈ *N* **and**
¬ *gr.INTERP M N* ⊫ *D* **and**
⋀*C*. *C* ∈ *N* ⟹ ¬ *gr.INTERP M N* ⊫ *C* ⟹ *D* ≤ *C*

then obtain *CAs AAs As E* **where**
cas-in: *set CAs* ⊆ *N* **and**
n-mod-cas: *gr.INTERP M N* ⊨*m mset CAs* **and**
ca-prod: ⋀*CA*. *CA* ∈ *set CAs* ⟹ *gr.production M N CA* ≠ {} **and**
e-res: *gr.ord-resolve M CAs D AAs As E* **and**
n-nmod-e: ¬ *gr.INTERP M N* ⊨ *E* **and**
e-lt-d: *E* < *D*
using *gr.ord-resolve-counterex-reducing* **by** *blast*
define ι **where**
ι = *Infer* (*CAs* @ [*D*]) *E*

have ι ∈ *G-Inf M*
unfolding ι*-def G-Inf-def* **using** *e-res* **by** *auto*
moreover have *prems-of* ι ≠ []
unfolding ι*-def* **by** *simp*
moreover have *main-prem-of* ι = *D*
unfolding ι*-def* **by** *simp*
moreover have *set* (*side-prems-of* ι) ⊆ *N*
unfolding ι*-def* **using** *cas-in* **by** *simp*
moreover have *gr.INTERP M N* ⊨*s set* (*side-prems-of* ι)
unfolding ι*-def* **using** *n-mod-cas ca-prod* **by** (*simp add*: *gr.productive-imp-INTERP true-clss-def*)
moreover have ¬ *gr.INTERP M N* ⊨ *concl-of* ι
unfolding ι*-def* **using** *n-nmod-e* **by** *simp*
moreover have *concl-of* ι < *D*
unfolding ι*-def* **using** *e-lt-d* **by** *simp*
ultimately show ∃ι ∈ *G-Inf M*. *prems-of* ι ≠ [] ∧ *main-prem-of* ι = *D* ∧ *set* (*side-prems-of* ι) ⊆ *N* ∧
gr.INTERP M N ⊨*s set* (*side-prems-of* ι) ∧ ¬ *gr.INTERP M N* ⊨ *concl-of* ι ∧ *concl-of* ι < *D*
by *blast*
qed

interpretation *G*: *calculus-with-standard-redundancy* ‹*G-Inf M*› ‹{{#}}› ‹(⊨∩*e*)›
‹(<) :: *'a clause* ⇒ *'a clause* ⇒ *bool*›
using *G-Inf-have-prems G-Inf-reductive*
by (*unfold-locales*) *simp-all*

interpretation *G*: *clausal-counterex-reducing-calculus-with-standard-redundancy G-Inf M*
gr.INTERP M
by (*unfold-locales*)

interpretation *G*: *Calculus.statically-complete-calculus* {{#}} *G-Inf M* (⊨∩*e*) *G.Red-I M G.Red-F*
by *unfold-locales* (*use G.clausal-saturated-complete* **in** *blast*)

sublocale *F*: *lifting-intersection F-Inf* {{#}} *UNIV G-Inf* λ*N*. (⊫∩*e*) *G.Red-I* λ*N*. *G.Red-F*
{{#}} λ*N*. *G-F G-I-opt* λ*D C C'*. *False*
proof (*unfold-locales*; (*intro ballI*)*?*)
show *UNIV* ≠ {}
by (*rule UNIV-not-empty*)
next
show *Calculus.consequence-relation* {{#}} (⊫∩*e*)
by (*fact consequence-relation-axioms*)
next
show ⋀*M*. *tiebreaker-lifting* {{#}} *F-Inf* {{#}} (⊫∩*e*) (*G-Inf M*) (*G.Red-I M*) *G.Red-F G-F* (*G-I-opt M*)
(λ*D C C'*. *False*)
proof
fix *M* ι
show *the* (*G-I-opt M* ι) ⊆ *G.Red-I M* (*G-F* (*concl-of* ι))
unfolding *option.sel*
proof
fix ι'
assume ι' ∈ *G-I M* ι
then obtain ϱ ϱ*s* **where**
ι'': ι' = *Infer* (*prems-of* ι ··*cl* ϱ*s*) (*concl-of* ι · ϱ) **and**
ϱ*-gr*: *is-ground-subst* ϱ **and**
ϱ*-infer*: *Infer* (*prems-of* ι ··*cl* ϱ*s*) (*concl-of* ι · ϱ) ∈ *G-Inf M*
unfolding *G-I-def* **by** *blast*

show ι' ∈ *G.Red-I M* (*G-F* (*concl-of* ι))
unfolding *G.Red-I-def G.redundant-infer-def mem-Collect-eq* **using** ι' ϱ*-gr* ϱ*-infer*
by (*metis Calculus.inference.sel*(*2*) *G-Inf-reductive empty-iff ground-subst-ground-cls grounding-of-cls-ground insert-iff subst-cls-eq-grounding-of-cls-subset-eq true-clss-union*)
qed
qed (*auto simp*: *G-F-def ex-ground-subst*)
qed

notation *F.entails-G* (**infix** ⊫∩*G e* *50*)

sublocale *F*: *sound-inference-system F-Inf* {{#}} (⊫∩*G e*)
proof
fix ι
assume *i-in*: ι ∈ *F-Inf*
moreover
{
fix *I* η
assume
I-entails-prems: ∀σ. *is-ground-subst* σ ⟶ *I* ⊫*s set* (*prems-of* ι) ·*cs* σ **and**
η*-gr*: *is-ground-subst* η
obtain *CAs AAs As* σ **where**

the-inf: *ord-resolve-rename S CAs* (*main-prem-of* ι) *AAs As* σ (*concl-of* ι)
and
CAs: *CAs* = *side-prems-of* ι
using *i-in* **unfolding** *F-Inf-def* **by** *auto*
have *prems*: *mset* (*prems-of* ι) = *mset* (*side-prems-of* ι) + {#*main-prem-of* ι#}
by (*metis* (*no-types*) *F-Inf-have-prems*[*OF i-in*] *add.right-neutral append-Cons append-Nil2*
append-butlast-last-id mset.simps(2) *mset-rev mset-single-iff-right rev-append rev-is-Nil-conv union-mset-add-mset-right*)
have *I* $\models$ *concl-of* ι · η
using *ord-resolve-rename-sound*[*OF the-inf*, *of I* η, *OF - η-gr*]
unfolding *CAs prems*[*symmetric*] **using** *I-entails-prems*
by (*metis set-mset-mset set-mset-subst-cls-mset-subst-clss true-clss-set-mset*)
}
ultimately show *set* (*inference.prems-of* ι) $\models\cap\mathcal{G}e$ {*concl-of* ι}
unfolding *F.entails-𝒢-def 𝒢-F-def true-Union-grounding-of-cls-iff* **by** *auto*
qed

lemma *F-stat-comp-calc*: ‹*Calculus.statically-complete-calculus* {{#}} *F-Inf* ($\models\cap\mathcal{G}e$) *F.Red-I-𝒢*
F.Red-F-𝒢-empty›
proof (*rule F.stat-ref-comp-to-non-ground-fam-inter*)
have $\bigwedge M$. *Calculus.statically-complete-calculus* {{#}} (*G-Inf M*) ($\models\cap e$) (*G.Red-I M*) *G.Red-F*
by (*fact G.statically-complete-calculus-axioms*)
then show ‹$\forall$ *q*∈*UNIV*. *Calculus.statically-complete-calculus* {{#}} (*G-Inf q*) ($\models\cap e$) (*G.Red-I q*) *G.Red-F*›
by *clarsimp*
next
fix *N*
assume *F.saturated N*
have *F.ground.Inf-from-q N* ($\bigcup$ (*𝒢-F* ‘ *N*)) ⊆ {ι. $\exists\,\iota' \in$ *F.Inf-from N*. $\iota \in$ *𝒢-I N* ι'}
∪ *G.Red-I N* ($\bigcup$ (*𝒢-F* ‘ *N*))
using *G-Inf-overapprox-F-Inf* **unfolding** *F.ground.Inf-from-q-def 𝒢-I-def* **by** *fastforce*
then show ‹$\exists$ *q*∈*UNIV*. *F.ground.Inf-from-q q* ($\bigcup$ (*𝒢-F* ‘ *N*))
⊆ {ι. $\exists\,\iota'$∈*F.Inf-from N*. *𝒢-I-opt q* $\iota' \neq$ *None* ∧ $\iota \in$ *the* (*𝒢-I-opt q* ι')} ∪ *G.Red-I q* ($\bigcup$ (*𝒢-F* ‘ *N*))›
by *auto*
qed

sublocale *F*: *Calculus.statically-complete-calculus* {{#}} *F-Inf* ($\models\cap\mathcal{G}e$) *F.Red-I-𝒢* *F.Red-F-𝒢-empty*
using *F-stat-comp-calc* **by** *blast*

sublocale *F'*: *Calculus.statically-complete-calculus* {{#}} *F-Inf* ($\models\cap\mathcal{G}e$) *F.empty-ord.Red-Red-I* *F.Red-F-𝒢-empty*

using *F.empty-ord.reduced-calc-is-calc F.empty-ord.stat-is-stat-red F-stat-comp-calc*
by *blast*

($\models\cap\mathcal{G}e$) is a conjunctive entailment, meaning that for $M \models\cap\mathcal{G}e\ N$ to hold, each clause in N must be entailed by M. Unfortunately, this clashes with requirement (D3) ⟦*Disjunctive-Consequence-Relations.consequence-relation ?bot ?entails*; $?M' \subseteq ?M$; $?N' \subseteq ?N$; *?entails ?M' ?N'*⟧ ⟹ *?entails ?M ?N* of a splitting calculus.

Therefore, we define a disjunctive version of this entailment by stating that $M \models\cup\mathcal{G}e\ N$ iff there is some $C \in N$ such that $M \models\cap\mathcal{G}e\ \{C\}$. This definition is not quite enough because it does not capture (D1) *Disjunctive-Consequence-Relations.consequence-relation ?bot ?entails* ⟹ *?entails* {*?bot*} {}. More specifically, if N is empty, then there does not exist a $C \in N$! But we know that $M \models\cup\mathcal{G}e\ \{\}$ if M is unsatisfiable. Hence $M \models\cup\mathcal{G}e\ N$ if M is unsatisfiable, or there exists some $C \in N$ such that $M \models\cap\mathcal{G}e\ \{C\}$. In addition, it is necessary to modify this definition to capture (D5) the compactness property of disjunctive entailment.

definition *entails-𝒢-disj* :: ‹*'a clause set* ⇒ *'a clause set* ⇒ *bool*› (**infix** ‹$\models\cup\mathcal{G}e$›
50) **where**
‹$M \models\cup\mathcal{G}e\ N \longleftrightarrow M \models\cap\mathcal{G}e\ \{\{\#\}\} \vee (\exists\ M' \subseteq M.\ \textit{finite}\ M' \wedge (\exists\ C \in N.\ M' \models\cap\mathcal{G}e\ \{C\}))$›

lemma *entails-𝒢-disj-subsets*: ‹$M' \subseteq M \Longrightarrow N' \subseteq N \Longrightarrow M' \models\cup\mathcal{G}e\ N' \Longrightarrow M \models\cup\mathcal{G}e\ N$›
by (*smt* (*verit*, *del-insts*) *F.entails-trans F.subset-entailed entails-𝒢-disj-def order-trans subsetD*)

lemma *entails-𝒢-disj-compactness*:
‹$M \models\cup\mathcal{G}e\ N \Longrightarrow \exists\ M'\ N'.\ M' \subseteq M \wedge N' \subseteq N \wedge \textit{finite}\ M' \wedge \textit{finite}\ N' \wedge M' \models\cup\mathcal{G}e\ N'$›
proof −
assume ‹$M \models\cup\mathcal{G}e\ N$›
then consider
(*M-unsat*) ‹$M \models\cap\mathcal{G}e\ \{\{\#\}\}$› |
(*b*) ‹$\exists\ M' \subseteq M.\ \textit{finite}\ M' \wedge (\exists\ C \in N.\ M' \models\cap\mathcal{G}e\ \{C\})$›
unfolding *entails-𝒢-disj-def*
by *blast*
then show *?thesis*
proof *cases*
case *M-unsat*

```
    then show ?thesis
      using unsat-G-compact[of M]
      unfolding entails-G-disj-def
      by blast
  next
    case b
    then show ?thesis
      unfolding entails-G-disj-def
     by (meson empty-subsetI finite.emptyI finite.insertI insert-subset subset-refl)
  qed
qed


lemma entails-G-disj-cut: ‹M ⟹∪G e N ∪ {C} ⟹ M' ∪ {C} ⟹∪G e N' ⟹ M
∪ M' ⟹∪G e N ∪ N'›
proof −
  assume M-entails-N-u-C: ‹M ⟹∪G e N ∪ {C}› and
         M'-u-C-entails-N': ‹M' ∪ {C} ⟹∪G e N'›
  then obtain P P' where
    P-subset-M: ‹P ⊆ M› and
    finite-P: ‹finite P› and
    P-entails-N-u-C: ‹P ⟹∪G e N ∪ {C}› and
    P'-subset-M'-u-C: ‹P' ⊆ M' ∪ {C}› and
    finite-P': ‹finite P'› and
    P'-entails-N': ‹P' ⟹∪G e N'›
    using entails-G-disj-compactness[OF M-entails-N-u-C]
          entails-G-disj-compactness[OF M'-u-C-entails-N'] entails-G-disj-subsets
    by blast

  have P-subset-M-u-M': ‹P ⊆ M ∪ M'›
    using P-subset-M
    by blast

  show ?thesis
  proof (cases ‹C ∈ P'›)
    case C-in-P': True

    define P'' where
      ‹P'' = P' − {C}›

    have P''-subset-M': ‹P'' ⊆ M'›
      using P'-subset-M'-u-C P''-def
      by blast

    have finite-P'': ‹finite P''›
      using finite-P' P''-def
      by blast

    consider
```

```
    (M-unsat) ‹P ⟙∩𝒢e {{#}}›
  | (M'-u-C-unsat) ‹P' ⟙∩𝒢e {{#}}›
  | (c) ‹(∃ C' ∈ N ∪ {C}. P ⟙∩𝒢e {C'}) ∧ (∃ C' ∈ N'. P' ⟙∩𝒢e {C'})›
    using P-entails-N-u-C P'-entails-N' finite-P finite-P'
    unfolding entails-𝒢-disj-def
    by (metis (no-types, lifting) F.entails-trans F.subset-entailed)
  then show ?thesis
  proof cases
    case M-unsat
    then have ‹P ⟙∪𝒢e N ∪ N'›
      using entails-𝒢-disj-def
      by blast
    then show ?thesis
        using entails-𝒢-disj-subsets[of P ‹M ∪ M'› ‹N ∪ N'› ‹N ∪ N'›, OF
P-subset-M-u-M']
      by blast
  next
    case M'-u-C-unsat
    then show ?thesis

    by (smt (z3) F.subset-entailed F-entails-𝒢-iff M-entails-N-u-C P'-subset-M'-u-C
UN-Un
        Un-insert-right entails-𝒢-disj-def entails-𝒢-disj-subsets insert-iff
        sup-bot.right-neutral sup-ge1 true-clss-union)
  next
    case c
    then obtain C1 C2 where
      C1-in-N-u-C: ‹C1 ∈ N ∪ {C}› and
      P-entails-C1: ‹P ⟙∩𝒢e {C1}› and
      C2-in-N': ‹C2 ∈ N'› and
      P'-entails-C2: ‹P' ⟙∩𝒢e {C2}›
      by blast
    then show ?thesis
    proof (cases ‹C1 = C›)
      case C1-is-C: True

      show ?thesis
      proof (cases ‹C2 = C›)
        case True
        then have ‹N ∪ {C} ∪ N' = N ∪ N'›
          using C2-in-N'
          by blast
        moreover have ‹P ⟙∪𝒢e N ∪ {C}›
          using P-entails-C1 C1-in-N-u-C finite-P
          unfolding entails-𝒢-disj-def
          by blast
        ultimately show ?thesis
          using entails-𝒢-disj-subsets[OF P-subset-M-u-M', of ‹N ∪ {C}› ‹N ∪
N'›]
```

```
        by blast
      next
        case C2-not-C: False
        then have ‹P ∪ P'' ╞═∩𝒢e {C2}›
       by (smt (verit, del-insts) C1-is-C F.entail-union F.entails-trans F.subset-entailed
              P''-def P'-entails-C2 P-entails-C1 Un-commute Un-insert-left in-
sert-Diff-single
            sup-ge2)
        then have ‹M ∪ M' ╞═∪𝒢e N'›
       using C2-in-N' P''-subset-M' P-subset-M finite-UnI[OF finite-P finite-P'']
          by (smt (verit, ccfv-SIG) P-subset-M-u-M' Un-subset-iff Un-upper2
entails-𝒢-disj-def
            order-trans)
        then show ?thesis
          by (meson entails-𝒢-disj-subsets equalityE sup-ge2)
      qed
    next
      case False
      then have ‹C1 ∈ N›
        using C1-in-N-u-C
        by blast
      then have ‹P ╞═∪𝒢e N›
        unfolding entails-𝒢-disj-def
        using P-entails-C1 finite-P
        by blast
      then show ?thesis
        using entails-𝒢-disj-subsets[OF P-subset-M-u-M']
        by blast
    qed
  qed
next
  case False
  then have ‹P' ⊆ M'›
    using P'-subset-M'-u-C
    by blast
  then have ‹M' ╞═∪𝒢e N'›
    using P'-entails-N' entails-𝒢-disj-subsets
    by blast
  then show ?thesis
    using entails-𝒢-disj-subsets[of M' ‹M ∪ M'› N' ‹N ∪ N'›]
    by blast
qed
qed

lemma entails-𝒢-disj-cons-rel-ext: ‹consequence-relation {#} (╞═∪𝒢e)›
proof (standard)
  show ‹{{#}} ╞═∪𝒢e {}›
    using F.subset-entailed entails-𝒢-disj-def
    by blast
```

show ‹$\bigwedge C.\ \{C\} \Vdash\cup\mathcal{G}e\ \{C\}$›
by (*meson F.subset-entailed entails-G-disj-def finite.emptyI finite.insertI singletonI subset-refl*)
show ‹$\bigwedge M'\ M\ N'\ N.\ M' \subseteq M \Longrightarrow N' \subseteq N \Longrightarrow M' \Vdash\cup\mathcal{G}e\ N' \Longrightarrow M \Vdash\cup\mathcal{G}e\ N$›
by (*rule entails-G-disj-subsets*)
show ‹$\bigwedge M\ N\ C\ M'\ N'.\ M \Vdash\cup\mathcal{G}e\ N \cup \{C\} \Longrightarrow M' \cup \{C\} \Vdash\cup\mathcal{G}e\ N' \Longrightarrow M \cup M' \Vdash\cup\mathcal{G}e\ N \cup N'$›
by (*rule entails-G-disj-cut*)
show ‹$\bigwedge M\ N.\ M \Vdash\cup\mathcal{G}e\ N \Longrightarrow \exists\ M'\ N'.\ M' \subseteq M \wedge N' \subseteq N \wedge$ *finite* $M' \wedge$ *finite* $N' \wedge M' \Vdash\cup\mathcal{G}e\ N'$›
by (*rule entails-G-disj-compactness*)
qed

sublocale *entails-G-disj-cons-rel*: *consequence-relation* ‹$\{\#\}$› ‹$(\Vdash\cup\mathcal{G}e)$›
by (*rule entails-G-disj-cons-rel-ext*)

notation *entails-G-disj-cons-rel.entails-neg* (**infix** ‹$\Vdash\cup\mathcal{G}e_\sim$› *50*)

lemma *all-redundant-to-bottom*: ‹$\mathcal{C} \neq \{\#\} \Longrightarrow \mathcal{C} \in$ *F.Red-F-G-empty* $\{\{\#\}\}$›
unfolding *F.Red-F-G-empty-def F.Red-F-G-empty-q-def G.Red-F-def*
proof −
assume *C-not-empty*: ‹$\mathcal{C} \neq \{\#\}$›
have ‹$D \in \mathcal{G}\text{-}F\ \mathcal{C} \Longrightarrow \exists\, DD \subseteq \{\{\#\}\}.\ (\forall\, I.\ I \models s\ DD \longrightarrow I \models D) \wedge (\forall\, Da \in DD.\ Da < D)$› **for** D
proof −
fix D :: ‹'*a clause*›

assume ‹$D \in \mathcal{G}\text{-}F\ \mathcal{C}$›
then have ‹$D \neq \{\#\}$›
using *C-not-empty* **unfolding** *G-F-def* **by** *force*
then have ‹$\{\#\} < D$›
by *auto*
moreover have ‹$\forall\ I.\ I \models s\ \{\{\#\}\} \longrightarrow I \models D$›
by *blast*
ultimately show ‹$\exists\ E \subseteq \{\{\#\}\}.\ (\forall\, I.\ I \models s\ E \longrightarrow I \models D) \wedge (\forall\ C \in E.\ C < D)$›
by *blast*
qed
then show ‹$\mathcal{C} \in (\bigcap q.\ \{C.\ \forall\, D \in \mathcal{G}\text{-}F\ C.\ D \in \{C.\ \exists\, DD \subseteq \bigcup\ (\mathcal{G}\text{-}F\ `\ \{\{\#\}\}).\ DD \models\cap e\ \{C\} \wedge (\forall\, D \in DD.\ D < C)\}\})$›
by *simp*
qed

lemma *bottom-never-redundant*: ‹$\{\#\} \notin$ *F.Red-F-G-empty* N›
unfolding *F.Red-F-G-empty-def F.Red-F-G-empty-q-def G.Red-F-def*
by *auto*

lemma ‹*F.Inf-between UNIV* (*F.Red-F-G-empty N*) ⊆ *F.empty-ord.Red-Red-I N*›
using *F.empty-ord.inf-subs-reduced-red-inf* **.**

end

9.6 Lightweight Avatar without BinSplit

Since the set 𝕡 of nullary predicates is left unspecified, we cannot define *fml* nor *asn*. Therefore, we keep them abstract and leave it to anybody instantiating this locale to specify them.

locale *LA-calculus* = *FO-resolution-prover-disjunctive S subst-atm id-subst comp-subst renaming-aparts*
atm-of-atms mgu less-atm
for
S :: ‹('*a* :: *wellorder*) *clause* ⇒ '*a clause*› **and**
subst-atm :: ‹'*a* ⇒ '*s* ⇒ '*a*› **and**
id-subst :: ‹'*s*› **and**
comp-subst :: ‹'*s* ⇒ '*s* ⇒ '*s*› **and**
renaming-aparts :: ‹'*a clause list* ⇒ '*s list*› **and**
atm-of-atms :: ‹'*a list* ⇒ '*a*› **and**
mgu :: ‹'*a set set* ⇒ '*s option*› **and**
less-atm :: ‹'*a* ⇒ '*a* ⇒ *bool*›
+ **fixes**
asn :: ‹'*a clause sign* ⇒ ('*v* :: *countable*) *sign set*› **and**
fml :: ‹'*v* ⇒ '*a clause*›
assumes
asn-not-empty: ‹*asn C* ≠ {}› **and**
fml-entails-C: ‹*a* ∈ *asn C* ⟹ {*map-sign fml a*} ⊨∪*G e*∼ {*C*}› **and**
C-entails-fml: ‹*a* ∈ *asn C* ⟹ {*C*} ⊨∪*G e*∼ {*map-sign fml a*}›
begin

interpretation *entails-G-disj-sound-inf-system*:
Calculi-And-Annotations.sound-inference-system F-Inf ‹{#}› ‹(⊨∪*G e*)›
proof *standard*
have ‹⋀ *ι*. *ι* ∈ *F-Inf* ⟹ *set* (*prems-of ι*) ⊨∩*G e* {*concl-of ι*}›
using *F.sound*
by *blast*
then show ‹⋀ *ι*. *ι* ∈ *F-Inf* ⟹ *set* (*prems-of ι*) ⊨∪*G e* {*concl-of ι*}›
using *entails-G-disj-def* **by** *blast*

qed

interpretation *LA-is-calculus*: *calculus* ‹{#}› *F-Inf* ‹(⊨∪*G e*)› *F.empty-ord.Red-Red-I F.Red-F-G-empty*
proof *standard*
show ‹⋀ *N*. *F.empty-ord.Red-Red-I N* ⊆ *F-Inf*›
using *F'.Red-I-to-Inf*
by *blast*
show ‹⋀ *N*. *N* ⊨∪*G e* {{#}} ⟹ *N* − *F.Red-F-G-empty N* ⊨∪*G e* {{#}}›

using *F.empty-ord.Red-F-Bot*
by (*metis* (*no-types*, *lifting*) *entails-$\mathcal{G}$-disj-def sat-$\mathcal{G}$-compact singleton-iff*)
show ‹$\bigwedge N\ N'.\ N \subseteq N' \Longrightarrow F.\text{Red-F-}\mathcal{G}\text{-empty}\ N \subseteq F.\text{Red-F-}\mathcal{G}\text{-empty}\ N'$›
using *F.empty-ord.Red-F-of-subset*
by *presburger*
show ‹$\bigwedge N\ N'.\ N \subseteq N' \Longrightarrow F.\text{empty-ord.Red-Red-I}\ N \subseteq F.\text{empty-ord.Red-Red-I}\ N'$›
using *F'.Red-I-of-subset*
by *presburger*
show ‹$\bigwedge N'\ N.\ N' \subseteq F.\text{Red-F-}\mathcal{G}\text{-empty}\ N \Longrightarrow F.\text{Red-F-}\mathcal{G}\text{-empty}\ N \subseteq F.\text{Red-F-}\mathcal{G}\text{-empty}\ (N - N')$›
using *F.empty-ord.Red-F-of-Red-F-subset*
by *blast*
show ‹$\bigwedge N'\ N.\ N' \subseteq F.\text{Red-F-}\mathcal{G}\text{-empty}\ N \Longrightarrow F.\text{empty-ord.Red-Red-I}\ N \subseteq F.\text{empty-ord.Red-Red-I}\ (N - N')$›
using *F'.Red-I-of-Red-F-subset*
by *presburger*
show ‹$\bigwedge \iota\ N.\ \iota \in F\text{-Inf} \Longrightarrow \text{concl-of}\ \iota \in N \Longrightarrow \iota \in F.\text{empty-ord.Red-Red-I}\ N$›
using *F'.Red-I-of-Inf-to-N*
by *blast*
qed

interpretation *LA-is-sound-calculus*: *sound-calculus* ‹$\{\#\}$› *F-Inf* ‹$(\Vdash_{\cup\mathcal{G}e})$› ‹$(\models_{\cup\mathcal{G}e})$›
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty
using *LA-is-calculus.Red-I-to-Inf LA-is-calculus.Red-F-Bot LA-is-calculus.Red-F-of-subset*
LA-is-calculus.Red-I-of-subset LA-is-calculus.Red-F-of-Red-F-subset
LA-is-calculus.Red-I-of-Red-F-subset LA-is-calculus.Red-I-of-Inf-to-N
by (*unfold-locales*, *presburger+*)

interpretation *LA-is-AF-calculus*: *calculus-with-annotated-consrel* ‹$\{\#\}$› *F-Inf* ‹$(\models_{\cup\mathcal{G}e})$› ‹$(\Vdash_{\cup\mathcal{G}e})$›
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty fml asn
proof *standard*
show ‹$\bigwedge C.\ \forall\ a \in asn\ C.\ \{map\text{-}sign\ fml\ a\} \Vdash_{\cup\mathcal{G}e_\sim} \{C\}$›
using *fml-entails-C*
by *blast*
show ‹$\bigwedge C.\ \forall\ a \in asn\ C.\ \{C\} \Vdash_{\cup\mathcal{G}e_\sim} \{map\text{-}sign\ fml\ a\}$›
using *C-entails-fml*
by *blast*
show ‹$\bigwedge C.\ asn\ C \neq \{\}$›
by (*rule asn-not-empty*)
qed

interpretation *core-LA-calculus*: *splitting-calculus* ‹$\{\#\}$› *F-Inf* ‹$(\Vdash_{\cup\mathcal{G}e})$› ‹$(\Vdash_{\cup\mathcal{G}e})$›
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty fml asn
proof *standard*

show ‹¬ {} ⊨∪$\mathcal{G}$e {}›
unfolding *entails-$\mathcal{G}$-disj-def* **using** *empty-not-unsat* **by** *blast*
show ‹⋀ *N*. *F.Inf-between UNIV* (*F.Red-F-$\mathcal{G}$-empty N*) ⊆ *F.empty-ord.Red-Red-I N*›
using *F.empty-ord.inf-subs-reduced-red-inf* **by** *blast*
show ‹⋀ *N*. {#} ∉ *F.Red-F-$\mathcal{G}$-empty N*›
using *bottom-never-redundant* **by** *blast*
show ‹⋀ $\mathcal{C}$. $\mathcal{C}$ ≠ {#} ⟹ $\mathcal{C}$ ∈ *F.Red-F-$\mathcal{G}$-empty* {{#}}›
using *all-redundant-to-bottom* **by** *blast*
qed

notation *LA-is-AF-calculus.AF-entails-sound* (**infix** ‹⊨s∪$\mathcal{G}$e$_{AF}$› *50*)
notation *LA-is-AF-calculus.AF-entails* (**infix** ‹⊨∪$\mathcal{G}$e$_{AF}$› *50*)

interpretation *AF-calculus-extended* ‹*to-AF* {#}›
core-LA-calculus.core.SInf ‹(⊨∪$\mathcal{G}$e$_{AF}$)› ‹(⊨s∪$\mathcal{G}$e$_{AF}$)› *core-LA-calculus.core.SRed$_I$*

core-LA-calculus.core.SRed$_F$ {} {}
using *core-LA-calculus.empty-simps.AF-calculus-extended-axioms* **.**

9.7 Lightweight Avatar

We now augment the earlier calculus into *LA* with the simplification rule BinSplit.

interpretation *with-BinSplit*: *AF-calculus-with-binsplit* ‹*to-AF* {#}› *core-LA-calculus.core.SInf LA-is-AF-calculus.AF-entails LA-is-AF-calculus.AF-entails-sound core-LA-calculus.core.SRed$_I$*

core-LA-calculus.core.SRed$_F$ ‹{}› ‹{}› *core-LA-calculus.bin-splittable*
using *core-LA-calculus.extend-simps-with-binsplit*[*OF core-LA-calculus.empty-simps.AF-calculus-extended-axioms*] **.**

sublocale *LA*: *AF-calculus-extended* ‹*to-AF* {#}›
core-LA-calculus.core.SInf LA-is-AF-calculus.AF-entails LA-is-AF-calculus.AF-entails-sound

core-LA-calculus.core.SRed$_I$ core-LA-calculus.core.SRed$_F$ with-BinSplit.Simps-with-BinSplit ‹{}›
using *with-BinSplit.AF-calc-ext.AF-calculus-extended-axioms* **.**

By Theorem ⟦*annotated-calculus ?bot ?FInf ?entails ?entails-sound ?FRed$_I$?FRed$_F$?fml ?asn*; *Calculi-And-Annotations.statically-complete-calculus ?bot ?FInf ?entails ?FRed$_I$?FRed$_F$*⟧ ⟹ *Calculi-And-Annotations.statically-complete-calculus* (*to-AF ?bot*) (*annotated-calculus.SInf ?bot ?FInf*) (*calculus-with-annotated-consrel.AF-entails ?entails*) (*annotated-calculus.SRed$_I$?bot ?FInf ?FRed$_I$*) (*annotated-calculus.SRed$_F$?FRed$_F$*), we can show that *LA* is statically complete, and therefore dynamically complete by Theorem ⟦*annotated-calculus ?bot ?FInf ?entails ?entails-sound ?FRed$_I$?FRed$_F$?fml ?asn*; *Calculi-And-Annotations.statically-complete-calculus ?bot ?FInf ?entails ?FRed$_I$?FRed$_F$*⟧ ⟹ *Calculi-And-Annotations.dynamically-complete-calculus* (*to-AF ?bot*) (*annotated-calculus.SInf ?bot ?FInf*) (*calculus-with-annotated-consrel.AF-entails*

?entails) (*annotated-calculus.SRed$_I$?bot ?FInf ?FRed$_I$*) (*annotated-calculus.SRed$_F$?FRed$_F$*).

lemma *F-disj-complete*: ‹*statically-complete-calculus* {#} *F-Inf* (⊫∪$\mathcal{G}$ *e*) *F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty*›
proof
show ‹⋀ *N*. *LA-is-calculus.saturated N* ⟹ *N* ⊫∪$\mathcal{G}$ *e* {{#}} ⟹ {#} ∈ *N*›
unfolding *LA-is-calculus.saturated-def* **using** *F′.saturated-def F′.statically-complete*
by (*smt* (*verit*, *ccfv-SIG*) *entails-$\mathcal{G}$-disj-def insertI1 sat-$\mathcal{G}$-compact singletonD*)
qed

theorem *strong-static-comp*:
‹*LA-is-AF-calculus.locally-saturated* $\mathcal{N}$ ⟹ $\mathcal{N}$ ⊨∪$\mathcal{G}$ *e$_{AF}$* {*to-AF* {#}} ⟹ *to-AF* {#} ∈ $\mathcal{N}$›
using *core-LA-calculus.core.S-calculus-strong-statically-complete*[*OF F-disj-complete*]
.

sublocale *strong-statically-complete-annotated-calculus* ‹{#}› *F-Inf* (⊫∪$\mathcal{G}$ *e*) (⊨∪$\mathcal{G}$ *e*)
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty fml asn core-LA-calculus.core.SInf core-LA-calculus.core.SRed$_I$ core-LA-calculus.core.SRed$_F$
using *strong-static-comp* **by** (*unfold-locales*, *blast*)

theorem *strong-dynamic-comp*: ‹*is-derivation core-LA-calculus.core.S-calculus.derive* $\mathcal{N}$ *i* ⟹
LA-is-AF-calculus.locally-fair $\mathcal{N}$ *i* ⟹ *llhd* $\mathcal{N}$ *i* ⊨∪$\mathcal{G}$ *e$_{AF}$* {*to-AF* {#}} ⟹
(∃ *i*. *to-AF* {#} ∈ *llnth* $\mathcal{N}$ *i i*)›
using *core-LA-calculus.core.S-calculus-strong-dynamically-complete*[*OF F-disj-complete*]
.

sublocale *strong-dynamically-complete-annotated-calculus* ‹{#}› *F-Inf* (⊫∪$\mathcal{G}$ *e*) (⊨∪$\mathcal{G}$ *e*)
F.empty-ord.Red-Red-I F.Red-F-$\mathcal{G}$-empty fml asn core-LA-calculus.core.SInf core-LA-calculus.core.SRed$_I$ core-LA-calculus.core.SRed$_F$
using *strong-dynamic-comp* **by** (*unfold-locales*, *blast*)

end

end

References

[1] G. Bergeron, F. Krasnopol, and S. Tourret. Formalizing splitting in isabelle/hol. In *ITP (to appear in)*, volume ?? of *LIPIcs*, page ?? Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[2] G. Ebner, J. Blanchette, and S. Tourret. A unifying splitting framework. In *CADE*, volume 12699 of *Lecture Notes in Computer Science*, pages 344–360. Springer, 2021.

[3] G. Ebner, J. Blanchette, and S. Tourret. Unifying splitting. *J. Autom. Reason.*, 67(2):16, 2023.