A Sound and Complete Calculus for Probability Inequalities

Matthew Doty

April 6, 2024

Abstract

We give a sound an complete multiple-conclusion calculus $\$ \vdash$ for finitely additive probability inequalities. In particular, we show

$$\sim\Gamma\$ \vdash \sim\Phi \equiv \forall \mathcal{P} \in \mathit{probabilities}. \sum \phi \leftarrow \Phi.\ \mathcal{P}\phi \leq \sum \gamma \leftarrow \Gamma.\ \mathcal{P}\gamma$$

…where $\sim \Gamma$ is the negation of all of the formulae in Γ (and similarly for $\sim \Phi$). We prove this by using an abstract form of *MaxSAT*. We also show

$$MaxSAT(\sim\Gamma \mathbin{@} \Phi)+c \leq \mathit{length}\,\Gamma \equiv \forall \mathcal{P} \in \mathit{probabilities}. \left(\sum \phi \leftarrow \Phi.\ \mathcal{P}\phi\right)+c \leq \sum \gamma \leftarrow \Gamma.\ \mathcal{P}\gamma$$

Finally, we establish a *collapse theorem*, which asserts that $(\sum \phi \leftarrow \Phi.\ \mathcal{P}\phi)+c \leq \sum \gamma \leftarrow \Gamma.\ \mathcal{P}\gamma$ holds for all probabilities $\mathcal{P}$ if and only if $(\sum \phi \leftarrow \Phi.\ \delta\phi)+c \leq \sum \gamma \leftarrow \Gamma.\ \delta\gamma$ holds for all binary-valued probabilities δ.

Contents

Chapter 1

Introduction

theory *Probability-Inequality-Completeness*
  imports
    Suppes-Theorem.Probability-Logic
begin

no-notation *FuncSet.funcset* (**infixr** $\rightarrow$ *60*)

We introduce a novel logical calculus and prove completeness for probability inequalities. This is a vast generalization of *Suppes' Theorem* which lays the foundation for this theory.

We provide two new logical judgements: *measure deduction* ($\$\vdash$) and *counting deduction* ($\#\vdash$). Both judgements capture a notion of *measure* or quantity. In both cases premises must be partially or completely *consumed* in sense to prove multiple conclusions. That is to say, a portion of the premises must be used to prove each conclusion which cannot be reused. Counting deduction counts the number of times a particular conclusion can be proved (as the name implies), while measure deduction includes multiple, different conclusions which must be proven via the premises.

We also introduce an abstract notion of MaxSAT, which is the maximal number of clauses in a list of clauses that can be simultaneously satisfied.

We show the following are equivalent:

- $\sim \Gamma \ \$\vdash \sim \Phi$
- $(\sim \Gamma \ @ \ \Phi) \ \#\vdash \ (\mathit{length} \ \Phi) \ \bot$
- $\mathit{MaxSAT} \ (\sim \Gamma \ @ \ \Phi) \leq \mathit{length} \ \Gamma$
- $\forall \ \delta \in \mathit{dirac\text{-}measures}. \ (\sum \varphi \leftarrow \Phi. \ \delta \ \varphi) \leq (\sum \gamma \leftarrow \Gamma. \ \delta \ \gamma)$
- $\forall \ \mathcal{P} \in \mathit{probabilities}. \ (\sum \varphi \leftarrow \Phi. \ \mathcal{P} \ \varphi) \leq (\sum \gamma \leftarrow \Gamma. \ \mathcal{P} \ \gamma)$

In the special case of MaxSAT, we show the following are equivalent:

- $\mathit{MaxSAT}\ (\sim \Gamma\ @\ \Phi) + c \leq \mathit{length}\ \Gamma$
- $\forall\ \delta \in \mathit{dirac\text{-}measures}.\ (\sum \varphi{\leftarrow}\Phi.\ \delta\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \delta\ \gamma)$
- $\forall\ \mathcal{P} \in \mathit{probabilities}.\ (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$

Chapter 2

Measure Deduction and Counting Deduction

2.1 Definition of Measure Deduction

To start, we introduce a common combinator for modifying functions that take two arguments.

definition *uncurry* :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a \times 'b \Rightarrow 'c$
  where *uncurry-def* [*simp*]: *uncurry* $f = (\lambda\ (x,\ y).\ f\ x\ y)$

Our new logical calculus is a recursively defined relation ($\$\vdash$) using *list deduction* ($:\vdash$).

We call our new logical relation *measure deduction*:

primrec (**in** *classical-logic*)
  measure-deduction :: *'a list* $\Rightarrow$ *'a list* $\Rightarrow$ *bool* (**infix** $\$\vdash$ *60*)
  where
    $\Gamma\ \$\vdash\ [] = True$
  $|\ \Gamma\ \$\vdash\ (\varphi\ \#\ \Phi) =$
      $(\exists\ \Psi.\ mset\ (map\ snd\ \Psi) \subseteq\#\ mset\ \Gamma$
            $\wedge\ map\ (uncurry\ (\sqcup))\ \Psi :\vdash \varphi$
            $\wedge\ map\ (uncurry\ (\rightarrow))\ \Psi\ @\ \Gamma \ominus (map\ snd\ \Psi)\ \$\vdash\ \Phi)$

Let us briefly analyze what the above definition is saying.

From the above we must find a special list-of-pairs Ψ, which we refer to as a *witness*, in order to establish $\Gamma\ \$\vdash\ \varphi\ \#\ \Phi$.

We may motivate measure deduction as follows. In the simplest case we know $\mathcal{P}\ \varphi \leq \mathcal{P}\ \psi + \Sigma$ if and only if $\mathcal{P}\ (\ \chi \sqcup \varphi\ ) + \mathcal{P}\ (\ \sim \chi \sqcup \varphi\ ) \leq \mathcal{P}\ \psi + \Sigma$, or equivalently $\mathcal{P}\ (\ \chi \sqcup \varphi\ ) + \mathcal{P}\ (\ \chi \rightarrow \varphi\ ) \leq \mathcal{P}\ \psi + \Sigma$. So it suffices to prove $\mathcal{P}\ (\ \chi \sqcup \varphi\ ) \leq \mathcal{P}\ \psi$ and $\mathcal{P}\ (\ \chi \rightarrow \varphi\ ) \leq \Sigma$. Here $[(\chi,\varphi)]$ is like the *witness* in our recursive definition, which reflects the $\exists\ \Psi.\ \ldots$ formula is our definition. The fact that measure deduction reflects proving theorems

in the theory of inequalities of probability logic is the elementary intuition behind the soundness theorem we will ultimately prove in §2.12.

A key difference from the simple motivation above is that, as in the case of Suppes' Theorem where we prove $\sim \Gamma :\vdash \sim \varphi$ if and only if $\mathcal{P}\ \varphi \leq (\sum\ \gamma \leftarrow \Gamma\ .\ \mathcal{P}\ \gamma)$ for all $\mathcal{P}$, soundness in this context means $\sim \Gamma\ \$\vdash \sim \Phi$ implies $\forall\ \mathcal{P}.\ (\sum \gamma \leftarrow \Gamma.\ \mathcal{P}\ \gamma) \geq (\sum \varphi \leftarrow \Phi.\ \mathcal{P}\ \varphi)$.

Another way of thinking about measure deduction is to think of Γ and Σ as bags of balls of soft clay and $\Gamma\ \$\vdash\ \Sigma$ meaning that we have shown Γ is *heavier than* Σ (ignoring, for the moment, that ($\$\vdash$) is not totally ordered). We have a scale ($:\vdash$) that lets us weigh several things on the left and one thing on the right at a time. We go through each clay ball σ in Σ one at a time without replacement, putting σ on the right of the scale. Then, we take a bunch of clay balls from Γ, cut them up as necessary (that is the $\psi \sqcup \gamma$ trick using the witness Ψ), and show they are heavier using our scale. We take the parts $\psi \to \gamma$ that we didn't use and put them back in our bag Γ. We will be able to reuse them later. If we can do this trick for every element σ in Σ successively using combinations of split leftovers in Γ, then we can show Γ is heavier than Σ (i.e., $\Gamma\ \$\vdash\ \Sigma$).

2.2 Definition of the Stronger Theory Relation

We next turn to looking at a subrelation of ($\$\vdash$), which we call the *stronger theory* relation ($\preceq$). Here we construe a *theory* as a list of propositions. We say theory Γ is *stronger than* Σ where, for each element σ in Σ, we can take an element γ of Γ *without replacement* such that $\vdash \gamma \to \sigma$.

To motivate this notion, let's reuse the metaphor that Γ and Σ are bags of balls of clay, and we need to show Γ is heavier without simply weighing the two bags. A sufficient (but incomplete) approach is to take each ball of clay σ in Σ and find another ball of clay γ in Γ (without replacement) that is heavier. This simple approach avoids the complexity of iteratively cutting up balls of clay.

definition (**in** *implication-logic*)
  stronger-theory-relation :: *'a list* ⇒ *'a list* ⇒ *bool* (**infix** $\preceq$ *100*)
  where
    $\Sigma \preceq \Gamma$ =
      (∃ Φ. *map snd* Φ = Σ
        ∧ *mset* (*map fst* Φ) ⊆# *mset* Γ
        ∧ (∀ (γ,σ) ∈ *set* Φ. ⊢ γ → σ))

abbreviation (**in** *implication-logic*)
  stronger-theory-relation-op :: *'a list* ⇒ *'a list* ⇒ *bool* (**infix** $\succeq$ *100*)
  where
    $\Gamma \succeq \Sigma \equiv \Sigma \preceq \Gamma$

2.3 The Stronger Theory Relation is a Preorder

Next, we show that $(\preceq)$ is a preorder by establishing reflexivity and transitivity.

We first prove the following lemma with respect to multisets and stronger theories.

lemma (**in** *implication-logic*) *msub-stronger-theory-intro*:
  assumes *mset* $\Sigma \subseteq\#$ *mset* Γ
  shows $\Sigma \preceq \Gamma$
proof −
  let *?*$\Delta\Sigma$ = *map* (λ *x*. (*x*,*x*)) Σ
  have *map snd ?*$\Delta\Sigma = \Sigma$
    by (*induct* Σ, *simp*, *simp*)
  moreover have *map fst ?*$\Delta\Sigma = \Sigma$
    by (*induct* Σ, *simp*, *simp*)
  hence *mset* (*map fst ?*$\Delta\Sigma$) $\subseteq\#$ *mset* Γ
    using *assms* **by** *simp*
  moreover have $\forall$ (γ,σ) $\in$ *set ?*$\Delta\Sigma$. $\vdash \gamma \rightarrow \sigma$
    by (*induct* Σ, *simp*, *simp*,
        metis list-implication.simps(1) list-implication-axiom-k)
  ultimately show *?thesis* **using** *stronger-theory-relation-def* **by** (*simp*, *blast*)
qed

The *reflexive* property immediately follows:

lemma (**in** *implication-logic*) *stronger-theory-reflexive* [*simp*]: $\Gamma \preceq \Gamma$
  using *msub-stronger-theory-intro* **by** *auto*

lemma (**in** *implication-logic*) *weakest-theory* [*simp*]: [] $\preceq \Gamma$
  using *msub-stronger-theory-intro* **by** *auto*

lemma (**in** *implication-logic*) *stronger-theory-empty-list-intro* [*simp*]:
  assumes $\Gamma \preceq$ []
  shows Γ = []
  using *assms stronger-theory-relation-def* **by** *simp*

Next, we turn to proving transitivity. We first prove two permutation theorems.

lemma (**in** *implication-logic*) *stronger-theory-right-permutation*:
  assumes $\Gamma \rightleftharpoons \Delta$
    and $\Sigma \preceq \Gamma$
  shows $\Sigma \preceq \Delta$
proof −
  from *assms(1)* **have** *mset* Γ = *mset* Δ
    by *simp*
  thus *?thesis*
    using *assms(2) stronger-theory-relation-def*
    by *fastforce*

qed

lemma (**in** *implication-logic*) *stronger-theory-left-permutation*:
assumes $\Sigma \rightleftharpoons \Delta$
and $\Sigma \preceq \Gamma$
shows $\Delta \preceq \Gamma$
proof −
have $\forall\ \Sigma\ \Gamma.\ \Sigma \rightleftharpoons \Delta \longrightarrow \Sigma \preceq \Gamma \longrightarrow \Delta \preceq \Gamma$
proof (*induct* Δ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* δ Δ)
{
fix Σ Γ
assume $\Sigma \rightleftharpoons (\delta \# \Delta)$ $\Sigma \preceq \Gamma$
from *this* **obtain** Φ **where** Φ:
map snd $\Phi = \Sigma$
mset (*map fst* Φ) $\subseteq\#$ *mset* Γ
$\forall\ (\gamma,\delta) \in$ *set* $\Phi.\ \vdash \gamma \rightarrow \delta$
using *stronger-theory-relation-def* **by** *fastforce*
with ‹$\Sigma \rightleftharpoons (\delta \# \Delta)$› **have** $\delta \in\#$ *mset* (*map snd* Φ)
by *fastforce*
from *this* **obtain** γ **where** γ: $(\gamma,\ \delta) \in\#$ *mset* Φ
by (*induct* Φ, *fastforce*+)
let *?*Φ_0 = *remove1* $(\gamma,\ \delta)$ Φ
let *?*Σ_0 = *map snd* *?*Φ_0
from γ Φ(*2*) **have** *mset* (*map fst* *?*Φ_0) $\subseteq\#$ *mset* (*remove1* γ Γ)
by (*metis ex-mset*
list-subtract-monotonic
list-subtract-mset-homomorphism
mset-remove1
remove1-pairs-list-projections-fst)
moreover have *mset* *?*Φ_0 $\subseteq\#$ *mset* Φ **by** *simp*
with Φ(*3*) **have** $\forall\ (\gamma,\delta) \in$ *set* *?*$\Phi_0.\ \vdash \gamma \rightarrow \delta$ **by** *fastforce*
ultimately have *?*$\Sigma_0 \preceq$ *remove1* γ Γ
unfolding *stronger-theory-relation-def* **by** *blast*
moreover have $\Delta \rightleftharpoons$ (*remove1* δ Σ) **using** ‹$\Sigma \rightleftharpoons (\delta \# \Delta)$›
by (*metis perm-remove-perm perm-sym remove-hd*)
moreover from γ Φ(*1*) **have** *mset* *?*Σ_0 = *mset* (*remove1* δ Σ)
using *remove1-pairs-list-projections-snd*
by *fastforce*
hence *?*$\Sigma_0 \rightleftharpoons$ *remove1* δ Σ
by *blast*
ultimately have $\Delta \preceq$ *remove1* γ Γ **using** *Cons*
by *presburger*
from *this* **obtain** Ψ_0 **where** Ψ_0:
map snd $\Psi_0 = \Delta$
mset (*map fst* Ψ_0) $\subseteq\#$ *mset* (*remove1* γ Γ)

```
    ∀ (γ,δ) ∈ set Ψ₀. ⊢ γ → δ
    using stronger-theory-relation-def by fastforce
  let ?Ψ = (γ, δ) # Ψ₀
  have map snd ?Ψ = (δ # Δ)
    by (simp add: Ψ₀(1))
  moreover have mset (map fst ?Ψ) ⊆# mset (γ # (remove1 γ Γ))
    using Ψ₀(2) by auto
  moreover from γ Φ(3) Ψ₀(3) have ∀ (γ,σ) ∈ set ?Ψ. ⊢ γ → σ by auto
  ultimately have (δ # Δ) ≼ (γ # (remove1 γ Γ))
    unfolding stronger-theory-relation-def by metis
  moreover from γ Φ(2) have γ ∈# mset Γ
    using mset-subset-eqD by fastforce
  hence (γ # (remove1 γ Γ)) ⇌ Γ
    by auto
  ultimately have (δ # Δ) ≼ Γ
    using stronger-theory-right-permutation by blast
 }
 then show ?case by blast
qed
with assms show ?thesis by blast
qed

lemma (in implication-logic) stronger-theory-transitive:
 assumes Σ ≼ Δ and Δ ≼ Γ
  shows Σ ≼ Γ
proof −
 have ∀ Δ Γ. Σ ≼ Δ ⟶ Δ ≼ Γ ⟶ Σ ≼ Γ
 proof (induct Σ)
  case Nil
  then show ?case using stronger-theory-relation-def by simp
 next
  case (Cons σ Σ)
  {
   fix Δ Γ
   assume (σ # Σ) ≼ Δ Δ ≼ Γ
   from this obtain Φ where Φ:
    map snd Φ = σ # Σ
    mset (map fst Φ) ⊆# mset Δ
    ∀ (δ,σ) ∈ set Φ. ⊢ δ → σ
    using stronger-theory-relation-def by (simp, metis)
   let ?δ = fst (hd Φ)
   from Φ(1) have Φ ≠ [] by (induct Φ, simp+)
   hence ?δ ∈# mset (map fst Φ) by (induct Φ, simp+)
   with Φ(2) have ?δ ∈# mset Δ by (meson mset-subset-eqD)
   hence mset (map fst (remove1 (hd Φ) Φ)) ⊆# mset (remove1 ?δ Δ)
    using ‹Φ ≠ []› Φ(2)
    by (simp,
        metis
         diff-single-eq-union
```

hd-in-set
image-mset-add-mset
insert-subset-eq-iff
set-mset-mset)
moreover have *remove1* (*hd* Φ) Φ = *tl* Φ
using ‹Φ ≠ []›
by (*induct* Φ, *simp+*)
moreover from Φ(*1*) **have** *map snd* (*tl* Φ) = Σ
by (*simp add*: *map-tl*)
moreover from Φ(*3*) **have** ∀ (δ,σ) ∈ *set* (*tl* Φ). ⊢ δ → σ
by (*simp add*: ‹Φ ≠ []› *list.set-sel*(*2*))
ultimately have Σ ⪯ *remove1* *?*δ Δ
using *stronger-theory-relation-def* **by** *auto*
from ‹*?*δ ∈# *mset* Δ› **have** *?*δ # (*remove1* *?*δ Δ) ⇌ Δ
by *fastforce*
with ‹Δ ⪯ Γ› **have** (*?*δ # (*remove1* *?*δ Δ)) ⪯ Γ
using *stronger-theory-left-permutation perm-sym* **by** *blast*
from *this* **obtain** Ψ **where** Ψ:
map snd Ψ = (*?*δ # (*remove1* *?*δ Δ))
mset (*map fst* Ψ) ⊆# *mset* Γ
∀ (γ,δ) ∈ *set* Ψ. ⊢ γ → δ
using *stronger-theory-relation-def* **by** (*simp*, *metis*)
let *?*γ = *fst* (*hd* Ψ)
from Ψ(*1*) **have** Ψ ≠ [] **by** (*induct* Ψ, *simp+*)
hence *?*γ ∈# *mset* (*map fst* Ψ) **by** (*induct* Ψ, *simp+*)
with Ψ(*2*) **have** *?*γ ∈# *mset* Γ **by** (*meson mset-subset-eqD*)
hence *mset* (*map fst* (*remove1* (*hd* Ψ) Ψ)) ⊆# *mset* (*remove1* *?*γ Γ)
using ‹Ψ ≠ []› Ψ(*2*)
by (*simp*,
metis
diff-single-eq-union
hd-in-set
image-mset-add-mset
insert-subset-eq-iff
set-mset-mset)
moreover from ‹Ψ ≠ []› **have** *remove1* (*hd* Ψ) Ψ = *tl* Ψ
by (*induct* Ψ, *simp+*)
moreover from Ψ(*1*) **have** *map snd* (*tl* Ψ) = (*remove1* *?*δ Δ)
by (*simp add*: *map-tl*)
moreover from Ψ(*3*) **have** ∀ (γ,δ) ∈ *set* (*tl* Ψ). ⊢ γ → δ
by (*simp add*: ‹Ψ ≠ []› *list.set-sel*(*2*))
ultimately have *remove1* *?*δ Δ ⪯ *remove1* *?*γ Γ
using *stronger-theory-relation-def* **by** *auto*
with ‹Σ ⪯ *remove1* *?*δ Δ› *Cons.hyps* **have** Σ ⪯ *remove1* *?*γ Γ
by *blast*
from *this* **obtain** Ω$_0$ **where** Ω$_0$:
map snd Ω$_0$ = Σ
mset (*map fst* Ω$_0$) ⊆# *mset* (*remove1* *?*γ Γ)
∀ (γ,σ) ∈ *set* Ω$_0$. ⊢ γ → σ

using *stronger-theory-relation-def* **by** (*simp*, *metis*)
let *?Ω* = (*?γ*, *σ*) # Ω_0
from Ω_0(*1*) **have** *map snd ?Ω* = *σ* # Σ **by** *simp*
moreover from Ω_0(*2*) **have** *mset* (*map fst ?Ω*) ⊆# *mset* (*?γ* # (*remove1 ?γ* Γ))
by *simp*
moreover from Φ(*1*) Ψ(*1*) **have** *σ* = *snd* (*hd* Φ) *?δ* = *snd* (*hd* Ψ) **by** *fastforce+*
with Φ(*3*) Ψ(*3*) ‹Φ ≠ []› ‹Ψ ≠ []› *hd-in-set* **have** ⊢ *?δ* → *σ* ⊢ *?γ* → *?δ*
by *fastforce+*
hence ⊢ *?γ* → *σ* **using** *modus-ponens hypothetical-syllogism* **by** *blast*
with Ω_0(*3*) **have** ∀ (*γ*,*σ*) ∈ *set ?Ω*. ⊢ *γ* → *σ*
by *auto*
ultimately have (*σ* # Σ) ⪯ (*?γ* # (*remove1 ?γ* Γ))
unfolding *stronger-theory-relation-def*
by *metis*
moreover from ‹*?γ* ∈# *mset* Γ› **have** (*?γ* # (*remove1 ?γ* Γ)) ⇌ Γ
by *force*
ultimately have (*σ* # Σ) ⪯ Γ
using *stronger-theory-right-permutation*
by *blast*
}
then show *?case* **by** *blast*
qed
thus *?thesis* **using** *assms* **by** *blast*
qed

2.4 The Stronger Theory Relation is a Subrelation of of Measure Deduction

Next, we show that Γ ⪰ Σ implies Γ $⊢ Σ. Before doing so we establish several helpful properties regarding the stronger theory relation (⪰).

lemma (**in** *implication-logic*) *stronger-theory-witness*:
assumes *σ* ∈ *set* Σ
shows Σ ⪯ Γ = (∃ *γ* ∈ *set* Γ. ⊢ *γ* → *σ* ∧ (*remove1 σ* Σ) ⪯ (*remove1 γ* Γ))
proof (*rule iffI*)
assume Σ ⪯ Γ
from *this* **obtain** Φ **where** Φ:
map snd Φ = Σ
mset (*map fst* Φ) ⊆# *mset* Γ
∀ (*γ*,*σ*) ∈ *set* Φ. ⊢ *γ* → *σ*
unfolding *stronger-theory-relation-def* **by** *blast*
from *assms* Φ(*1*) **obtain** *γ* **where** *γ*: (*γ*, *σ*) ∈# *mset* Φ
by (*induct* Φ, *fastforce+*)
hence *γ* ∈# *mset* (*map fst* Φ) **by** *force*
hence *γ* ∈# *mset* Γ **using** Φ(*2*)
by (*meson mset-subset-eqD*)

moreover
let *?*Φ_0 = *remove1* (γ, σ) Φ
let *?*Σ_0 = *map snd ?*Φ_0
from γ Φ(*2*) **have** *mset* (*map fst ?*Φ_0) $\subseteq\#$ *mset* (*remove1* γ Γ)
by (*metis*
ex-mset
list-subtract-monotonic
list-subtract-mset-homomorphism
remove1-pairs-list-projections-fst
mset-remove1)
moreover have *mset ?*Φ_0 $\subseteq\#$ *mset* Φ **by** *simp*
with Φ(*3*) **have** $\forall$ (γ,σ) $\in$ *set ?*Φ_0. $\vdash$ γ $\rightarrow$ σ **by** *fastforce*
ultimately have *?*Σ_0 $\preceq$ *remove1* γ Γ
unfolding *stronger-theory-relation-def* **by** *blast*
moreover from γ Φ(*1*) **have** *mset ?*Σ_0 = *mset* (*remove1* σ Σ)
using *remove1-pairs-list-projections-snd*
by *fastforce*
hence *?*Σ_0 $\rightleftharpoons$ *remove1* σ Σ
by *linarith*
ultimately have *remove1* σ Σ $\preceq$ *remove1* γ Γ
using *stronger-theory-left-permutation*
by *blast*
moreover from γ Φ(*3*) **have** $\vdash$ γ $\rightarrow$ σ **by** (*simp*, *fast*)
moreover from γ Φ(*2*) **have** γ $\in\#$ *mset* Γ
using *mset-subset-eqD* **by** *fastforce*
ultimately show $\exists$ γ $\in$ *set* Γ. $\vdash$ γ $\rightarrow$ σ $\wedge$ (*remove1* σ Σ) $\preceq$ (*remove1* γ Γ) **by** *auto*
next
assume $\exists$ γ $\in$ *set* Γ. $\vdash$ γ $\rightarrow$ σ $\wedge$ (*remove1* σ Σ) $\preceq$ (*remove1* γ Γ)
from *this* **obtain** Φ γ **where** γ: γ $\in$ *set* Γ $\vdash$ γ $\rightarrow$ σ
and Φ: *map snd* Φ = (*remove1* σ Σ)
mset (*map fst* Φ) $\subseteq\#$ *mset* (*remove1* γ Γ)
$\forall$ (γ,σ) $\in$ *set* Φ. $\vdash$ γ $\rightarrow$ σ
unfolding *stronger-theory-relation-def* **by** *blast*
let *?*Φ = (γ, σ) # Φ
from Φ(*1*) **have** *map snd ?*Φ = σ # (*remove1* σ Σ) **by** *simp*
moreover from Φ(*2*) γ(*1*) **have** *mset* (*map fst ?*Φ) $\subseteq\#$ *mset* Γ
by (*simp add*: *insert-subset-eq-iff*)
moreover from Φ(*3*) γ(*2*) **have** $\forall$ (γ,σ) $\in$ *set ?*Φ. $\vdash$ γ $\rightarrow$ σ
by *auto*
ultimately have (σ # (*remove1* σ Σ)) $\preceq$ Γ
unfolding *stronger-theory-relation-def* **by** *metis*
moreover from *assms* **have** σ # (*remove1* σ Σ) $\rightleftharpoons$ Σ
by *force*
ultimately show Σ $\preceq$ Γ
using *stronger-theory-left-permutation* **by** *blast*
qed

lemma (**in** *implication-logic*) *stronger-theory-cons-witness*:

```
  (σ # Σ) ≼ Γ = (∃ γ ∈ set Γ. ⊢ γ → σ ∧ Σ ≼ (remove1 γ Γ))
proof −
  have σ ∈# mset (σ # Σ) by simp
  hence (σ # Σ) ≼ Γ = (∃ γ ∈ set Γ. ⊢ γ → σ ∧ (remove1 σ (σ # Σ)) ≼ (remove1
γ Γ))
    by (meson list.set-intros(1) stronger-theory-witness)
  thus ?thesis by simp
qed

lemma (in implication-logic) stronger-theory-left-cons:
  assumes (σ # Σ) ≼ Γ
  shows Σ ≼ Γ
proof −
  from assms obtain Φ where Φ:
    map snd Φ = σ # Σ
    mset (map fst Φ) ⊆# mset Γ
    ∀ (δ,σ) ∈ set Φ. ⊢ δ → σ
    using stronger-theory-relation-def by (simp, metis)
  let ?Φ′ = remove1 (hd Φ) Φ
  from Φ(1) have map snd ?Φ′ = Σ by (induct Φ, simp+)
  moreover from Φ(2) have mset (map fst ?Φ′) ⊆# mset Γ
    by (metis diff-subset-eq-self
              list-subtract.simps(1)
              list-subtract.simps(2)
              list-subtract-mset-homomorphism
              map-monotonic
              subset-mset.dual-order.trans)
  moreover from Φ(3) have ∀ (δ,σ) ∈ set ?Φ′. ⊢ δ → σ by fastforce
  ultimately show ?thesis unfolding stronger-theory-relation-def by blast
qed

lemma (in implication-logic) stronger-theory-right-cons:
  assumes Σ ≼ Γ
  shows Σ ≼ (γ # Γ)
proof −
  from assms obtain Φ where Φ:
    map snd Φ = Σ
    mset (map fst Φ) ⊆# mset Γ
    ∀(γ, σ)∈set Φ. ⊢ γ → σ
    unfolding stronger-theory-relation-def
    by auto
  hence mset (map fst Φ) ⊆# mset (γ # Γ)
    by (metis Diff-eq-empty-iff-mset
              list-subtract.simps(2)
              list-subtract-mset-homomorphism
              mset-zero-iff remove1.simps(1))
  with Φ(1) Φ(3) show ?thesis
    unfolding stronger-theory-relation-def
    by auto
```

qed

lemma (**in** *implication-logic*) *stronger-theory-left-right-cons*:
assumes $\vdash \gamma \to \sigma$
and $\Sigma \preceq \Gamma$
shows $(\sigma \# \Sigma) \preceq (\gamma \# \Gamma)$
proof −
from *assms*(*2*) **obtain** Φ **where** Φ:
map snd $\Phi = \Sigma$
mset (*map fst* Φ) $\subseteq\#$ *mset* Γ
$\forall (\gamma, \sigma) \in set\ \Phi. \vdash \gamma \to \sigma$
unfolding *stronger-theory-relation-def*
by *auto*
let *?*$\Phi = (\gamma, \sigma) \# \Phi$
from *assms*(*1*) Φ **have**
map snd ?$\Phi = \sigma \# \Sigma$
mset (*map fst ?*Φ) $\subseteq\#$ *mset* $(\gamma \# \Gamma)$
$\forall (\gamma, \sigma) \in set\ ?\Phi. \vdash \gamma \to \sigma$
by *fastforce*+
thus *?thesis*
unfolding *stronger-theory-relation-def*
by *metis*
qed

lemma (**in** *implication-logic*) *stronger-theory-relation-alt-def*:
$\Sigma \preceq \Gamma = (\exists \Phi.\ mset\ (map\ snd\ \Phi) = mset\ \Sigma \wedge$
$mset\ (map\ fst\ \Phi) \subseteq\#\ mset\ \Gamma \wedge$
$(\forall (\gamma, \sigma) \in set\ \Phi. \vdash \gamma \to \sigma))$
proof (*induct* Γ *arbitrary*: Σ)
case *Nil*
then show *?case*
using *stronger-theory-empty-list-intro*
stronger-theory-reflexive
by (*simp*, *blast*)
next
case (*Cons* γ Γ)
have $\Sigma \preceq (\gamma \# \Gamma) = (\exists \Phi.\ mset\ (map\ snd\ \Phi) = mset\ \Sigma \wedge$
$mset\ (map\ fst\ \Phi) \subseteq\#\ mset\ (\gamma \# \Gamma) \wedge$
$(\forall (\gamma, \sigma) \in set\ \Phi. \vdash \gamma \to \sigma))$
proof (*rule iffI*)
assume $\Sigma \preceq (\gamma \# \Gamma)$
thus $\exists \Phi.\ mset\ (map\ snd\ \Phi) = mset\ \Sigma \wedge$
$mset\ (map\ fst\ \Phi) \subseteq\#\ mset\ (\gamma \# \Gamma) \wedge$
$(\forall (\gamma, \sigma) \in set\ \Phi. \vdash \gamma \to \sigma)$
unfolding *stronger-theory-relation-def*
by *metis*
next
assume $\exists \Phi.\ mset\ (map\ snd\ \Phi) = mset\ \Sigma \wedge$
$mset\ (map\ fst\ \Phi) \subseteq\#\ mset\ (\gamma \# \Gamma) \wedge$

$(\forall\,(\gamma,\,\sigma){\in}\mathit{set}\ \Phi.\ \vdash \gamma \rightarrow \sigma)$
from *this* **obtain** Φ **where** Φ:
mset (*map snd* Φ) = *mset* Σ
mset (*map fst* Φ) $\subseteq\#$ *mset* (γ # Γ)
$\forall\,(\gamma,\,\sigma){\in}\mathit{set}\ \Phi.\ \vdash \gamma \rightarrow \sigma$
by *metis*
show $\Sigma \preceq (\gamma\ \#\ \Gamma)$
proof (*cases* $\exists\ \sigma.\ (\gamma,\,\sigma) \in \mathit{set}\ \Phi$)
assume $\exists\ \sigma.\ (\gamma,\,\sigma) \in \mathit{set}\ \Phi$
from *this* **obtain** σ **where** σ: $(\gamma,\,\sigma) \in \mathit{set}\ \Phi$ **by** *auto*
let *?*Φ = *remove1* $(\gamma,\,\sigma)$ Φ
from σ **have** *mset* (*map snd ?*Φ) = *mset* (*remove1* σ Σ)
using Φ(*1*) *remove1-pairs-list-projections-snd* **by** *force+*
moreover
from σ **have** *mset* (*map fst ?*Φ) = *mset* (*remove1* γ (*map fst* Φ))
using Φ(*1*) *remove1-pairs-list-projections-fst* **by** *force+*
with Φ(*2*) **have** *mset* (*map fst ?*Φ) $\subseteq\#$ *mset* Γ
by (*simp add*: *subset-eq-diff-conv*)
moreover from Φ(*3*) **have** $\forall\,(\gamma,\,\sigma){\in}\mathit{set}\ ?\Phi.\ \vdash \gamma \rightarrow \sigma$
by *fastforce*
ultimately have *remove1* σ $\Sigma \preceq \Gamma$ **using** *Cons* **by** *blast*
from *this* **obtain** Ψ **where** Ψ:
map snd Ψ = *remove1* σ Σ
mset (*map fst* Ψ) $\subseteq\#$ *mset* Γ
$\forall\,(\gamma,\,\sigma){\in}\mathit{set}\ \Psi.\ \vdash \gamma \rightarrow \sigma$
unfolding *stronger-theory-relation-def*
by *blast*
let *?*Ψ = $(\gamma,\,\sigma)$ # Ψ
from Ψ **have** *map snd ?*Ψ = σ # (*remove1* σ Σ)
mset (*map fst ?*Ψ) $\subseteq\#$ *mset* (γ # Γ)
by *simp+*
moreover from Φ(*3*) σ **have** $\vdash \gamma \rightarrow \sigma$ **by** *auto*
with Ψ(*3*) **have** $\forall\,(\gamma,\,\sigma){\in}\mathit{set}\ ?\Psi.\ \vdash \gamma \rightarrow \sigma$ **by** *auto*
ultimately have (σ # (*remove1* σ Σ)) $\preceq$ (γ # Γ)
unfolding *stronger-theory-relation-def*
by *metis*
moreover
have $\sigma \in \mathit{set}\ \Sigma$
by (*metis* Φ(*1*) σ *set-mset-mset set-zip-rightD zip-map-fst-snd*)
hence $\Sigma \rightleftharpoons \sigma$ # (*remove1* σ Σ)
by *auto*
hence $\Sigma \preceq (\sigma$ # (*remove1* σ Σ))
using *stronger-theory-reflexive*
stronger-theory-right-permutation
by *blast*
ultimately show *?thesis*
using *stronger-theory-transitive*
by *blast*
next

```
      assume ∄σ. (γ, σ) ∈ set Φ
      hence γ ∉ set (map fst Φ) by fastforce
      with Φ(2) have mset (map fst Φ) ⊆# mset Γ
        by (metis diff-single-trivial
                  in-multiset-in-set
                  insert-DiffM2
                  mset-remove1
                  remove-hd
                  subset-eq-diff-conv)
      hence Σ ≼ Γ
        using Cons Φ(1) Φ(3)
        by blast
      thus ?thesis
        using stronger-theory-right-cons
        by auto
    qed
  qed
  thus ?case by auto
qed

lemma (in implication-logic) stronger-theory-deduction-monotonic:
  assumes Σ ≼ Γ
      and Σ :⊢ φ
    shows Γ :⊢ φ
using assms
proof (induct Σ arbitrary: φ)
  case Nil
  then show ?case
    by (simp add: list-deduction-weaken)
next
  case (Cons σ Σ)
  assume (σ # Σ) ≼ Γ (σ # Σ) :⊢ φ
  hence Σ :⊢ σ → φ Σ ≼ Γ
    using
      list-deduction-theorem
      stronger-theory-left-cons
    by (blast, metis)
  with Cons have Γ :⊢ σ → φ by blast
  moreover
  have σ ∈ set (σ # Σ) by auto
  with ‹(σ # Σ) ≼ Γ› obtain γ where γ: γ ∈ set Γ ⊢ γ → σ
    using stronger-theory-witness by blast
  hence Γ :⊢ σ
    using
      list-deduction-modus-ponens
      list-deduction-reflection
      list-deduction-weaken
    by blast
  ultimately have Γ :⊢ φ
```

```
    using list-deduction-modus-ponens by blast
  then show ?case by blast
qed

lemma (in classical-logic) measure-msub-left-monotonic:
  assumes mset Σ ⊆# mset Γ
      and Σ $⊢ Φ
    shows Γ $⊢ Φ
  using assms
proof (induct Φ arbitrary: Σ Γ)
  case Nil
  then show ?case by simp
next
  case (Cons φ Φ)
  from this obtain Ψ where Ψ:
    mset (map snd Ψ) ⊆# mset Σ
    map (uncurry (⊔)) Ψ :⊢ φ
    map (uncurry (→)) Ψ @ Σ ⊖ (map snd Ψ) $⊢ Φ
    using measure-deduction.simps(2) by blast
  let ?Ψ = map snd Ψ
  let ?Ψ' = map (uncurry (→)) Ψ
  let ?Σ' = ?Ψ' @ (Σ ⊖ ?Ψ)
  let ?Γ' = ?Ψ' @ (Γ ⊖ ?Ψ)
  from Ψ have mset ?Ψ ⊆# mset Γ
    using ‹mset Σ ⊆# mset Γ› subset-mset.trans by blast
  moreover have mset (Σ ⊖ ?Ψ) ⊆# mset (Γ ⊖ ?Ψ)
    by (metis ‹mset Σ ⊆# mset Γ› list-subtract-monotonic)
  hence mset ?Σ' ⊆# mset ?Γ'
    by simp
  with Cons.hyps Ψ(3) have ?Γ' $⊢ Φ by blast
  ultimately have Γ $⊢ (φ # Φ)
    using Ψ(2) by fastforce
  then show ?case
    by simp
qed

lemma (in classical-logic) witness-weaker-theory:
  assumes mset (map snd Σ) ⊆# mset Γ
  shows map (uncurry (⊔)) Σ ≼ Γ
proof −
  have ∀ Γ. mset (map snd Σ) ⊆# mset Γ ⟶ map (uncurry (⊔)) Σ ≼ Γ
  proof (induct Σ)
    case Nil
    then show ?case by simp
  next
    case (Cons σ Σ)
    {
      fix Γ
      assume mset (map snd (σ # Σ)) ⊆# mset Γ
```

```
    hence mset (map snd Σ) ⊆# mset (remove1 (snd σ) Γ)
      by (simp add: insert-subset-eq-iff)
    with Cons have map (uncurry (⊔)) Σ ≼ remove1 (snd σ) Γ by blast
    moreover have uncurry (⊔) = (λ σ. fst σ ⊔ snd σ) by fastforce
    hence uncurry (⊔) σ = fst σ ⊔ snd σ by simp
    moreover have ⊢ snd σ → (fst σ ⊔ snd σ)
      unfolding disjunction-def
      by (simp add: axiom-k)
    ultimately have map (uncurry (⊔)) (σ # Σ) ≼ (snd σ # (remove1 (snd σ)
Γ))
      by (simp add: stronger-theory-left-right-cons)
    moreover have mset (snd σ # (remove1 (snd σ) Γ)) = mset Γ
      using ‹mset (map snd (σ # Σ)) ⊆# mset Γ›
      by (simp, meson insert-DiffM mset-subset-eq-insertD)
    ultimately have map (uncurry (⊔)) (σ # Σ) ≼ Γ
      unfolding stronger-theory-relation-alt-def
      by simp
  }
  then show ?case by blast
 qed
 with assms show ?thesis by simp
qed

lemma (in implication-logic) stronger-theory-combine:
 assumes Φ ≼ Δ
   and Ψ ≼ Γ
  shows (Φ @ Ψ) ≼ (Δ @ Γ)
proof −
 have ∀ Φ. Φ ≼ Δ ⟶ (Φ @ Ψ) ≼ (Δ @ Γ)
 proof (induct Δ)
  case Nil
  then show ?case
   using assms(2) stronger-theory-empty-list-intro by fastforce
 next
  case (Cons δ Δ)
  {
   fix Φ
   assume Φ ≼ (δ # Δ)
   from this obtain Σ where Σ:
    map snd Σ = Φ
    mset (map fst Σ) ⊆# mset (δ # Δ)
    ∀ (δ,φ) ∈ set Σ. ⊢ δ → φ
    unfolding stronger-theory-relation-def
    by blast
   have (Φ @ Ψ) ≼ ((δ # Δ) @ Γ)
   proof (cases ∃ φ . (δ, φ) ∈ set Σ)
    assume ∃ φ . (δ, φ) ∈ set Σ
    from this obtain φ where φ: (δ, φ) ∈ set Σ by auto
    let ?Σ = remove1 (δ, φ) Σ
```

from φ $\Sigma(1)$ **have** *mset* (*map snd* ?Σ) = *mset* (*remove1* φ Φ)
using *remove1-pairs-list-projections-snd* **by** *fastforce*
moreover from φ **have** *mset* (*map fst* ?Σ) = *mset* (*remove1* δ (*map fst* Σ))
using *remove1-pairs-list-projections-fst* **by** *fastforce*
hence *mset* (*map fst* ?Σ) $\subseteq\#$ *mset* Δ
using $\Sigma(2)$ *mset.simps*(*1*) *subset-eq-diff-conv* **by** *force*
moreover from $\Sigma(3)$ **have** $\forall$ (δ,φ) $\in$ *set* ?Σ. $\vdash$ $\delta \rightarrow \varphi$ **by** *auto*
ultimately have *remove1* φ $\Phi \preceq \Delta$
unfolding *stronger-theory-relation-alt-def* **by** *blast*
hence (*remove1* φ Φ @ Ψ) $\preceq$ (Δ @ Γ) **using** *Cons* **by** *auto*
from *this* **obtain** Ω **where** Ω:
map snd Ω = (*remove1* φ Φ) @ Ψ
mset (*map fst* Ω) $\subseteq\#$ *mset* (Δ @ Γ)
$\forall$ (α,β) $\in$ *set* Ω. $\vdash$ $\alpha \rightarrow \beta$
unfolding *stronger-theory-relation-def*
by *blast*
let ?Ω = (δ, φ) # Ω
have *map snd* ?Ω = φ # *remove1* φ Φ @ Ψ
using $\Omega(1)$ **by** *simp*
moreover have *mset* (*map fst* ?Ω) $\subseteq\#$ *mset* ((δ # Δ) @ Γ)
using $\Omega(2)$ **by** *simp*
moreover have $\vdash$ $\delta \rightarrow \varphi$
using $\Sigma(3)$ φ **by** *blast*
hence $\forall$ (α,β) $\in$ *set* ?Ω. $\vdash$ $\alpha \rightarrow \beta$ **using** $\Omega(3)$ **by** *auto*
ultimately have (φ # *remove1* φ Φ @ Ψ) $\preceq$ ((δ # Δ) @ Γ)
by (*metis stronger-theory-relation-def*)
moreover have φ $\in$ *set* Φ
using $\Sigma(1)$ φ **by** *force*
hence (φ # *remove1* φ Φ) $\rightleftharpoons$ Φ
by *force*
hence (φ # *remove1* φ Φ @ Ψ) $\rightleftharpoons$ Φ @ Ψ
by (*metis append-Cons perm-append2*)
ultimately show *?thesis*
using *stronger-theory-left-permutation* **by** *blast*
next
assume $\nexists\varphi$. (δ, φ) $\in$ *set* Σ
hence $\delta \notin$ *set* (*map fst* Σ)
mset Δ + *add-mset* δ (*mset* []) = *mset* (δ # Δ)
by *auto*
hence *mset* (*map fst* Σ) $\subseteq\#$ *mset* Δ
by (*metis* (*no-types*) ‹*mset* (*map fst* Σ) $\subseteq\#$ *mset* (δ # Δ)›
diff-single-trivial
mset.simps(*1*)
set-mset-mset
subset-eq-diff-conv)
with $\Sigma(1)$ $\Sigma(3)$ **have** $\Phi \preceq \Delta$
unfolding *stronger-theory-relation-def*
by *blast*
hence (Φ @ Ψ) $\preceq$ (Δ @ Γ) **using** *Cons* **by** *auto*

```
      then show ?thesis
        by (simp add: stronger-theory-right-cons)
    qed
  }
  then show ?case by blast
 qed
 thus ?thesis using assms by blast
qed
```

We now turn to proving that $(\succeq)$ is a subrelation of $(:\vdash)$.

```
lemma (in classical-logic) stronger-theory-to-measure-deduction:
 assumes Γ ⪰ Σ
 shows Γ $⊢ Σ
proof −
 have ∀ Γ. Σ ≼ Γ ⟶ Γ $⊢ Σ
 proof (induct Σ)
  case Nil
  then show ?case by fastforce
 next
  case (Cons σ Σ)
  {
   fix Γ
   assume (σ # Σ) ≼ Γ
   from this obtain γ where γ: γ ∈ set Γ ⊢ γ → σ Σ ≼ (remove1 γ Γ)
    using stronger-theory-cons-witness by blast
   let ?Φ = [(γ,γ)]
   from γ Cons have (remove1 γ Γ) $⊢ Σ by blast
   moreover have mset (remove1 γ Γ) ⊆# mset (map (uncurry (→)) ?Φ @ Γ
⊖ (map snd ?Φ))
    by simp
   ultimately have map (uncurry (→)) ?Φ @ Γ ⊖ (map snd ?Φ) $⊢ Σ
    using measure-msub-left-monotonic by blast
   moreover have map (uncurry (⊔)) ?Φ :⊢ σ
    by (simp, metis γ(2)
                    Peirces-law
                    disjunction-def
                    list-deduction-def
                    list-deduction-modus-ponens
                    list-deduction-weaken
                    list-implication.simps(1)
                    list-implication.simps(2))
   moreover from γ(1) have mset (map snd ?Φ) ⊆# mset Γ by simp
   ultimately have Γ $⊢ (σ # Σ)
    using measure-deduction.simps(2) by blast
  }
  then show ?case by blast
 qed
 thus ?thesis using assms by blast
qed
```

2.5 Measure Deduction is a Preorder

We next show that measure deduction is a preorder.

Reflexivity follows immediately because $(\preceq)$ is a subrelation and is itself reflexive.

theorem (**in** *classical-logic*) *measure-reflexive*: Γ \$⊢ Γ
  by (*simp add*: *stronger-theory-to-measure-deduction*)

Transitivity is complicated. It requires constructing many witnesses and involves a lot of metatheorems. Below we provide various witness constructions that allow us to establish $[\![\Gamma \$\vdash \Lambda; \Lambda \$\vdash \Delta]\!] \Longrightarrow \Gamma \$\vdash \Delta$.

primrec (**in** *implication-logic*)
  first-component :: $('a \times 'a)$ *list* ⇒ $('a \times 'a)$ *list* ⇒ $('a \times 'a)$ *list* $(\mathfrak{A})$
  where
    $\mathfrak{A}\ \Psi\ [] = []$
  | $\mathfrak{A}\ \Psi\ (\delta\ \#\ \Delta) =$
      (*case find* $(\lambda\ \psi.\ (uncurry\ (\rightarrow))\ \psi = snd\ \delta)\ \Psi$ *of*
          None ⇒ $\mathfrak{A}\ \Psi\ \Delta$
        | *Some* ψ ⇒ $\psi\ \#\ (\mathfrak{A}\ (remove1\ \psi\ \Psi)\ \Delta)$)

primrec (**in** *implication-logic*)
  second-component :: $('a \times 'a)$ *list* ⇒ $('a \times 'a)$ *list* ⇒ $('a \times 'a)$ *list* $(\mathfrak{B})$
  where
    $\mathfrak{B}\ \Psi\ [] = []$
  | $\mathfrak{B}\ \Psi\ (\delta\ \#\ \Delta) =$
      (*case find* $(\lambda\ \psi.\ (uncurry\ (\rightarrow))\ \psi = snd\ \delta)\ \Psi$ *of*
          None ⇒ $\mathfrak{B}\ \Psi\ \Delta$
        | *Some* ψ ⇒ $\delta\ \#\ (\mathfrak{B}\ (remove1\ \psi\ \Psi)\ \Delta)$)

lemma (**in** *implication-logic*) *first-component-second-component-mset-connection*:
  mset (*map* (*uncurry* $(\rightarrow)$) $(\mathfrak{A}\ \Psi\ \Delta)$) = *mset* (*map snd* $(\mathfrak{B}\ \Psi\ \Delta)$)
proof −
  have ∀ Ψ. *mset* (*map* (*uncurry* $(\rightarrow)$) $(\mathfrak{A}\ \Psi\ \Delta)$) = *mset* (*map snd* $(\mathfrak{B}\ \Psi\ \Delta)$)
  proof (*induct* Δ)
    case *Nil*
    then show *?case* **by** *simp*
  next
    case (*Cons* $\delta\ \Delta$)
    {
      fix Ψ
      have *mset* (*map* (*uncurry* $(\rightarrow)$) $(\mathfrak{A}\ \Psi\ (\delta\ \#\ \Delta))$) =
            mset (*map snd* $(\mathfrak{B}\ \Psi\ (\delta\ \#\ \Delta))$)
      proof (*cases find* $(\lambda\ \psi.\ (uncurry\ (\rightarrow))\ \psi = snd\ \delta)\ \Psi = None$)
        case *True*
        then show *?thesis* **using** *Cons* **by** *simp*
      next
        case *False*

```
      from this obtain ψ where
        find (λψ. uncurry (→) ψ = snd δ) Ψ = Some ψ
        uncurry (→) ψ = snd δ
        using find-Some-predicate
        by fastforce
      then show ?thesis using Cons by simp
    qed
  }
  then show ?case by blast
 qed
 thus ?thesis by blast
qed

lemma (in implication-logic) second-component-right-empty [simp]:
 𝔅 [] Δ = []
 by (induct Δ, simp+)

lemma (in implication-logic) first-component-msub:
 mset (𝔄 Ψ Δ) ⊆# mset Ψ
proof −
 have ∀ Ψ. mset (𝔄 Ψ Δ) ⊆# mset Ψ
 proof(induct Δ)
  case Nil
  then show ?case by simp
 next
  case (Cons δ Δ)
  {
   fix Ψ
   have mset (𝔄 Ψ (δ # Δ)) ⊆# mset Ψ
   proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
    case True
    then show ?thesis using Cons by simp
   next
    case False
    from this obtain ψ where
     ψ: find (λψ. uncurry (→) ψ = snd δ) Ψ = Some ψ
        ψ ∈ set Ψ
     using find-Some-set-membership
     by fastforce
    have mset (𝔄 (remove1 ψ Ψ) Δ) ⊆# mset (remove1 ψ Ψ)
     using Cons by metis
    thus ?thesis using ψ by (simp add: insert-subset-eq-iff)
   qed
  }
  then show ?case by blast
 qed
 thus ?thesis by blast
qed
```

```
lemma (in implication-logic) second-component-msub:
  mset (𝔅 Ψ Δ) ⊆# mset Δ
proof −
  have ∀ Ψ. mset (𝔅 Ψ Δ) ⊆# mset Δ
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have mset (𝔅 Ψ (δ # Δ)) ⊆# mset (δ # Δ)
      using Cons
      by (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None,
          simp,
          metis add-mset-remove-trivial
                diff-subset-eq-self
                subset-mset.order-trans,
          auto)
    }
    thus ?case by blast
  qed
  thus ?thesis by blast
qed

lemma (in implication-logic) second-component-snd-projection-msub:
  mset (map snd (𝔅 Ψ Δ)) ⊆# mset (map (uncurry (→)) Ψ)
proof −
  have ∀ Ψ. mset (map snd (𝔅 Ψ Δ)) ⊆# mset (map (uncurry (→)) Ψ)
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have mset (map snd (𝔅 Ψ (δ # Δ))) ⊆# mset (map (uncurry (→)) Ψ)
      proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
        case True
        then show ?thesis
          using Cons by simp
      next
        case False
        from this obtain ψ where ψ:
          find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = Some ψ
          by auto
        hence 𝔅 Ψ (δ # Δ) = δ # (𝔅 (remove1 ψ Ψ) Δ)
          using ψ by fastforce
        with Cons have mset (map snd (𝔅 Ψ (δ # Δ))) ⊆#
```

```
                mset ((snd δ) # map (uncurry (→)) (remove1 ψ Ψ))
      by (simp, metis mset-map mset-remove1)
    moreover from ψ have snd δ = (uncurry (→)) ψ
      using find-Some-predicate by fastforce
    ultimately have
      mset (map snd (𝔅 Ψ (δ # Δ))) ⊆#
        mset (map (uncurry (→)) (ψ # (remove1 ψ Ψ)))
      by simp
    thus ?thesis
      by (metis
          first-component-msub
          first-component-second-component-mset-connection
          map-monotonic)
  qed
 }
 thus ?case by blast
qed
thus ?thesis by blast
qed

lemma (in implication-logic) second-component-diff-msub:
 assumes mset (map snd Δ) ⊆# mset (map (uncurry (→)) Ψ @ Γ ⊖ (map snd
Ψ))
 shows mset (map snd (Δ ⊖ (𝔅 Ψ Δ))) ⊆# mset (Γ ⊖ (map snd Ψ))
proof −
 have ∀ Ψ Γ. mset (map snd Δ) ⊆# mset (map (uncurry (→)) Ψ @ Γ ⊖ (map
snd Ψ)) ⟶
          mset (map snd (Δ ⊖ (𝔅 Ψ Δ))) ⊆# mset (Γ ⊖ (map snd Ψ))
 proof (induct Δ)
  case Nil
  then show ?case by simp
 next
  case (Cons δ Δ)
  {
   fix Ψ Γ
   assume ♢: mset (map snd (δ # Δ)) ⊆# mset (map (uncurry (→)) Ψ @ Γ
⊖ map snd Ψ)
   have mset (map snd ((δ # Δ) ⊖ 𝔅 Ψ (δ # Δ))) ⊆# mset (Γ ⊖ map snd Ψ)
   proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
    case True
    hence A: snd δ ∉ set (map (uncurry (→)) Ψ)
    proof (induct Ψ)
     case Nil
     then show ?case by simp
    next
     case (Cons ψ Ψ)
     then show ?case
      by (cases uncurry (→) ψ = snd δ, simp+)
    qed
```

moreover have
mset (*map snd* Δ)
⊆# *mset* (*map* (*uncurry* (→)) Ψ @ Γ ⊖ *map snd* Ψ) − {#*snd* δ#}
using ♢ *insert-subset-eq-iff* **by** *fastforce*
ultimately have
mset (*map snd* Δ)
⊆# *mset* (*map* (*uncurry* (→)) Ψ @ (*remove1* (*snd* δ) Γ)
⊖ *map snd* Ψ)
by (*metis* (*no-types*)
mset-remove1
union-code
list-subtract.simps(*2*)
list-subtract-remove1-cons-perm
remove1-append)
hence *B*: *mset* (*map snd* (Δ ⊖ (𝔅 Ψ Δ))) ⊆# *mset* (*remove1* (*snd* δ) Γ ⊖ (*map snd* Ψ))
using *Cons* **by** *blast*
have *C*: *snd* δ ∈# *mset* (*snd* δ # *map snd* Δ @
(*map* (*uncurry* (→)) Ψ @ Γ ⊖ *map snd* Ψ) ⊖ (*snd* δ # *map snd* Δ))
by (*meson in-multiset-in-set list.set-intros*(*1*))
have *mset* (*map snd* (δ # Δ))
+ (*mset* (*map* (*uncurry* (→)) Ψ @ Γ ⊖ *map snd* Ψ)
− *mset* (*map snd* (δ # Δ)))
= *mset* (*map* (*uncurry* (→)) Ψ @ Γ ⊖ *map snd* Ψ)
using ♢ *subset-mset.add-diff-inverse* **by** *blast*
then have *snd* δ ∈# *mset* (*map* (*uncurry* (→)) Ψ) + (*mset* Γ − *mset* (*map snd* Ψ))
using *C* **by** *simp*
with *A* **have** *snd* δ ∈ *set* Γ
by (*metis* (*no-types*) *diff-subset-eq-self*
in-multiset-in-set
subset-mset.add-diff-inverse
union-iff)
have *D*: 𝔅 Ψ Δ = 𝔅 Ψ (δ # Δ)
using ‹*find* (λψ. *uncurry* (→) ψ = *snd* δ) Ψ = *None*›
by *simp*
obtain *diff* :: *'a list* ⇒ *'a list* ⇒ *'a list* **where**
∀ *x0 x1*. (∃ *v2*. *x1* @ *v2* ⇌ *x0*) = (*x1* @ *diff x0 x1* ⇌ *x0*)
by *moura*
then have *E*:
mset (*map snd* (𝔅 Ψ (δ # Δ))
@ *diff* (*map* (*uncurry* (→)) Ψ) (*map snd* (𝔅 Ψ (δ # Δ))))
= *mset* (*map* (*uncurry* (→)) Ψ)
by (*meson second-component-snd-projection-msub mset-le-perm-append*)
have *F*: ∀ *a m ma*. (*add-mset* (*a*::*'a*) *m* ⊆# *ma*) = (*a* ∈# *ma* ∧ *m* ⊆# *ma* − {#*a*#})
using *insert-subset-eq-iff* **by** *blast*
then have *snd* δ ∈# *mset* (*map snd* (𝔅 Ψ (δ # Δ))

```
                                @ diff (map (uncurry (→)) Ψ) (map snd (𝔅 Ψ (δ #
Δ))))
                          + mset (Γ ⊖ map snd Ψ)
      using E ◇ by force
    then have snd δ ∈# mset (Γ ⊖ map snd Ψ)
      using A E by (metis (no-types) in-multiset-in-set union-iff)
    then have G: add-mset (snd δ) (mset (map snd (Δ ⊖ 𝔅 Ψ Δ))) ⊆# mset
(Γ ⊖ map snd Ψ)
      using B F by force
    have H: ∀ ps psa f. ¬ mset (ps::('a × 'a) list) ⊆# mset psa ∨
                        mset ((map f psa::'a list) ⊖ map f ps) = mset (map f (psa
⊖ ps))
      using map-list-subtract-mset-equivalence by blast
    have snd δ ∉# mset (map snd (𝔅 Ψ (δ # Δ)))
                + mset (diff (map (uncurry (→)) Ψ) (map snd (𝔅 Ψ (δ # Δ))))
      using A E by auto
    then have add-mset (snd δ) (mset (map snd (Δ ⊖ 𝔅 Ψ Δ)))
          = mset (map snd (δ # Δ) ⊖ map snd (𝔅 Ψ (δ # Δ)))
      using D H second-component-msub by auto
    then show ?thesis
      using G H by (metis (no-types) second-component-msub)
  next
    case False
   from this obtain ψ where ψ: find (λψ. uncurry (→) ψ = snd δ) Ψ = Some
ψ
      by auto
    let ?Ψ' = remove1 ψ Ψ
    let ?Γ' = remove1 (snd ψ) Γ
    have snd δ = uncurry (→) ψ
        ψ ∈ set Ψ
        mset ((δ # Δ) ⊖ 𝔅 Ψ (δ # Δ)) =
         mset (Δ ⊖ 𝔅 ?Ψ' Δ)
      using ψ find-Some-predicate find-Some-set-membership
      by fastforce+
    moreover
    have mset (Γ ⊖ map snd Ψ) = mset (?Γ' ⊖ map snd ?Ψ')
        by (simp, metis ‹ψ ∈ set Ψ› image-mset-add-mset in-multiset-in-set
insert-DiffM)
    moreover
    obtain search :: ('a × 'a) list ⇒ ('a × 'a ⇒ bool) ⇒ 'a × 'a where
      ∀ xs P. (∃ x. x ∈ set xs ∧ P x) = (search xs P ∈ set xs ∧ P (search xs P))
      by moura
    then have ∀ p ps. (find p ps ≠ None ∨ (∀ pa. pa ∉ set ps ∨ ¬ p pa))
                  ∧ (find p ps = None ∨ search ps p ∈ set ps ∧ p (search ps p))
      by (metis (full-types) find-None-iff)
    then have (find (λp. uncurry (→) p = snd δ) Ψ ≠ None
            ∨ (∀ p. p ∉ set Ψ ∨ uncurry (→) p ≠ snd δ))
          ∧ (find (λp. uncurry (→) p = snd δ) Ψ = None
            ∨ search Ψ (λp. uncurry (→) p = snd δ) ∈ set Ψ
```

```
            ∧ uncurry (→) (search Ψ (λp. uncurry (→) p = snd δ)) = snd δ)
      by blast
    hence snd δ ∈ set (map (uncurry (→)) Ψ)
      by (metis (no-types) False image-eqI image-set)
    moreover
    have A: add-mset (uncurry (→) ψ) (image-mset snd (mset Δ))
        = image-mset snd (add-mset δ (mset Δ))
      by (simp add: ‹snd δ = uncurry (→) ψ›)
    have B: {#snd δ#} ⊆# image-mset (uncurry (→)) (mset Ψ)
      using ‹snd δ ∈ set (map (uncurry (→)) Ψ)› by force
    have image-mset (uncurry (→)) (mset Ψ) − {#snd δ#}
      = image-mset (uncurry (→)) (mset (remove1 ψ Ψ))
      by (simp add: ‹ψ ∈ set Ψ› ‹snd δ = uncurry (→) ψ› image-mset-Diff)
    then have mset (map snd (Δ ⊖ 𝔅 (remove1 ψ Ψ) Δ))
          ⊆# mset (remove1 (snd ψ) Γ ⊖ map snd (remove1 ψ Ψ))
      by (metis (no-types)
            A B ♢ Cons.hyps
            calculation(1)
            calculation(4)
            insert-subset-eq-iff
            mset.simps(2)
            mset-map
            subset-mset.diff-add-assoc2
            union-code)
    ultimately show ?thesis by fastforce
  qed
 }
 then show ?case by blast
qed
thus ?thesis using assms by auto
qed

primrec (in classical-logic)
 merge-witness :: ('a × 'a) list ⇒ ('a × 'a) list ⇒ ('a × 'a) list (𝔍)
 where
  𝔍 Ψ [] = Ψ
| 𝔍 Ψ (δ # Δ) =
    (case find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ of
       None ⇒ δ # 𝔍 Ψ Δ
     | Some ψ ⇒ (fst δ ⊓ fst ψ, snd ψ) # (𝔍 (remove1 ψ Ψ) Δ))

lemma (in classical-logic) merge-witness-right-empty [simp]:
 𝔍 [] Δ = Δ
 by (induct Δ, simp+)

lemma (in classical-logic) second-component-merge-witness-snd-projection:
 mset (map snd Ψ @ map snd (Δ ⊖ (𝔅 Ψ Δ))) = mset (map snd (𝔍 Ψ Δ))
proof −
 have ∀ Ψ. mset (map snd Ψ @ map snd (Δ ⊖ (𝔅 Ψ Δ))) = mset (map snd (𝔍
```

```
Ψ Δ))
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have mset (map snd Ψ @ map snd ((δ # Δ) ⊖ 𝔅 Ψ (δ # Δ))) =
           mset (map snd (𝔍 Ψ (δ # Δ)))
      proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
        case True
        then show ?thesis
          using Cons
          by (simp,
              metis (no-types, lifting)
                    ab-semigroup-add-class.add-ac(1)
                    add-mset-add-single
                    image-mset-single
                    image-mset-union
                    second-component-msub
                    subset-mset.add-diff-assoc2)
      next
        case False
       from this obtain ψ where ψ: find (λψ. uncurry (→) ψ = snd δ) Ψ = Some
ψ
          by auto
        moreover have ψ ∈ set Ψ
          by (meson ψ find-Some-set-membership)
        moreover
        let ?Ψ' = remove1 ψ Ψ
        from Cons have
          mset (map snd ?Ψ' @ map snd (Δ ⊖ 𝔅 ?Ψ' Δ)) =
            mset (map snd (𝔍 ?Ψ' Δ))
          by blast
        ultimately show ?thesis
          by (simp,
              metis (no-types, lifting)
                    add-mset-remove-trivial-eq
                    image-mset-add-mset
                    in-multiset-in-set
                    union-mset-add-mset-left)
      qed
    }
    then show ?case by blast
  qed
  thus ?thesis by blast
qed
```

```
lemma (in classical-logic) second-component-merge-witness-stronger-theory:
  (map (uncurry (→)) Δ @ map (uncurry (→)) Ψ ⊖ map snd (𝔅 Ψ Δ)) ≼
    map (uncurry (→)) (𝔍 Ψ Δ)
proof −
  have ∀ Ψ. (map (uncurry (→)) Δ @
             map (uncurry (→)) Ψ ⊖ map snd (𝔅 Ψ Δ)) ≼
             map (uncurry (→)) (𝔍 Ψ Δ)
  proof (induct Δ)
    case Nil
    then show ?case
      by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have ⊢ (uncurry (→)) δ → (uncurry (→)) δ
        using axiom-k modus-ponens implication-absorption by blast
      have
        (map (uncurry (→)) (δ # Δ) @
          map (uncurry (→)) Ψ ⊖ map snd (𝔅 Ψ (δ # Δ))) ≼
          map (uncurry (→)) (𝔍 Ψ (δ # Δ))
      proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
        case True
        thus ?thesis
          using Cons
                ‹⊢ (uncurry (→)) δ → (uncurry (→)) δ›
          by (simp, metis stronger-theory-left-right-cons)
      next
        case False
       from this obtain ψ where ψ: find (λψ. uncurry (→) ψ = snd δ) Ψ = Some
ψ
          by auto
        from ψ have snd δ = uncurry (→) ψ
          using find-Some-predicate by fastforce
        from ψ ‹snd δ = uncurry (→) ψ› have
          mset (map (uncurry (→)) (δ # Δ) @
                  map (uncurry (→)) Ψ ⊖ map snd (𝔅 Ψ (δ # Δ))) =
           mset (map (uncurry (→)) (δ # Δ) @
                  map (uncurry (→)) (remove1 ψ Ψ) ⊖
                  map snd (𝔅 (remove1 ψ Ψ) Δ))
          by (simp add: find-Some-set-membership image-mset-Diff)
        hence
          (map (uncurry (→)) (δ # Δ) @
             map (uncurry (→)) Ψ ⊖ map snd (𝔅 Ψ (δ # Δ))) ≼
           (map (uncurry (→)) (δ # Δ) @
             map (uncurry (→)) (remove1 ψ Ψ) ⊖ map snd (𝔅 (remove1 ψ Ψ) Δ))
          by (simp add: msub-stronger-theory-intro)
        with Cons ‹⊢ (uncurry (→)) δ → (uncurry (→)) δ› have
          (map (uncurry (→)) (δ # Δ) @
```

```
        map (uncurry (→)) Ψ ⊖ map snd (𝔅 Ψ (δ # Δ)))
        ≼ ((uncurry (→)) δ # map (uncurry (→)) (𝔍 (remove1 ψ Ψ) Δ))
      using stronger-theory-left-right-cons
          stronger-theory-transitive
      by fastforce
    moreover
    let ?α = fst δ
    let ?β = fst ψ
    let ?γ = snd ψ
    have uncurry (→) = (λ δ. fst δ → snd δ) by fastforce
    with ψ have (uncurry (→)) δ = ?α → ?β → ?γ
      using find-Some-predicate by fastforce
    hence ⊢ ((?α ⊓ ?β) → ?γ) → (uncurry (→)) δ
      using biconditional-def curry-uncurry by auto
    with ψ have
      ((uncurry (→)) δ # map (uncurry (→)) (𝔍 (remove1 ψ Ψ) Δ)) ≼
       map (uncurry (→)) (𝔍 Ψ (δ # Δ))
      using stronger-theory-left-right-cons by auto
    ultimately show ?thesis
      using stronger-theory-transitive
      by blast
  qed
 }
 then show ?case by simp
qed
thus ?thesis by simp
qed

lemma (in classical-logic) merge-witness-msub-intro:
 assumes mset (map snd Ψ) ⊆# mset Γ
    and mset (map snd Δ) ⊆# mset (map (uncurry (→)) Ψ @ Γ ⊖ (map snd
Ψ))
  shows mset (map snd (𝔍 Ψ Δ)) ⊆# mset Γ
proof −
 have ∀ Ψ Γ. mset (map snd Ψ) ⊆# mset Γ ⟶
          mset (map snd Δ) ⊆# mset (map (uncurry (→)) Ψ @ Γ ⊖ (map snd
Ψ)) ⟶
          mset (map snd (𝔍 Ψ Δ)) ⊆# mset Γ
 proof (induct Δ)
  case Nil
  then show ?case by simp
 next
  case (Cons δ Δ)
  {
   fix Ψ :: ('a × 'a) list
   fix Γ :: 'a list
   assume ♢: mset (map snd Ψ) ⊆# mset Γ
          mset (map snd (δ # Δ)) ⊆# mset (map (uncurry (→)) Ψ @ Γ ⊖
(map snd Ψ))
```

have *mset* (*map snd* (ℑ Ψ (*δ* # Δ))) ⊆# *mset* Γ
proof (*cases find* (λ *ψ*. (*uncurry* (→)) *ψ* = *snd δ*) Ψ = *None*)
case *True*
hence *snd δ* ∉ *set* (*map* (*uncurry* (→)) Ψ)
proof (*induct* Ψ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons ψ* Ψ)
hence *uncurry* (→) *ψ* ≠ *snd δ* **by** *fastforce*
with *Cons* **show** *?case* **by** *fastforce*
qed
with ♢(*2*) **have** *snd δ* ∈# *mset* (Γ ⊖ *map snd* Ψ)
using *mset-subset-eq-insertD* **by** *fastforce*
with ♢(*1*) **have** *mset* (*map snd* Ψ) ⊆# *mset* (*remove1* (*snd δ*) Γ)
by (*metis list-subtract-mset-homomorphism*
mset-remove1
single-subset-iff
subset-mset.add-diff-assoc
subset-mset.add-diff-inverse
subset-mset.le-iff-add)
moreover
have *add-mset* (*snd δ*) (*mset* (Γ ⊖ *map snd* Ψ) − {#*snd δ*#}) = *mset* (Γ ⊖ *map snd* Ψ)
by (*meson* ‹*snd δ* ∈# *mset* (Γ ⊖ *map snd* Ψ)› *insert-DiffM*)
then have *image-mset snd* (*mset* Δ) − (*mset* Γ − *add-mset* (*snd δ*) (*image-mset snd* (*mset* Ψ)))
⊆# {#*x* → *y*. (*x*, *y*) ∈# *mset* Ψ#}
using ♢(*2*) **by** (*simp*, *metis add-mset-diff-bothsides*
list-subtract-mset-homomorphism
mset-map subset-eq-diff-conv)
hence *mset* (*map snd* Δ)
⊆# *mset* (*map* (*uncurry* (→)) Ψ @ (*remove1* (*snd δ*) Γ) ⊖ (*map snd* Ψ))
using *subset-eq-diff-conv* **by** (*simp*, *blast*)
ultimately have *mset* (*map snd* (ℑ Ψ Δ)) ⊆# *mset* (*remove1* (*snd δ*) Γ)
using *Cons* **by** *blast*
hence *mset* (*map snd* (*δ* # (ℑ Ψ Δ))) ⊆# *mset* Γ
by (*simp*, *metis* ‹*snd δ* ∈# *mset* (Γ ⊖ *map snd* Ψ)›
cancel-ab-semigroup-add-class.diff-right-commute
diff-single-trivial
insert-subset-eq-iff
list-subtract-mset-homomorphism
multi-drop-mem-not-eq)
with ‹*find* (λ *ψ*. (*uncurry* (→)) *ψ* = *snd δ*) Ψ = *None*›
show *?thesis*
by *simp*
next
case *False*
from *this* **obtain** *ψ* **where** *ψ*:

find ($\lambda\psi$. *uncurry* ($\rightarrow$) ψ = *snd* δ) Ψ = *Some* ψ
by *fastforce*
let *?*χ = *fst* ψ
let *?*γ = *snd* ψ
have *uncurry* ($\rightarrow$) = (λ ψ. *fst* ψ $\rightarrow$ *snd* ψ)
by *fastforce*
moreover
from *this* **have** *uncurry* ($\rightarrow$) ψ = *?*χ $\rightarrow$ *?*γ **by** *fastforce*
with ψ **have** *A*: (*?*χ, *?*γ) $\in$ *set* Ψ
and *B*: *snd* δ = *?*χ $\rightarrow$ *?*γ
using *find-Some-predicate*
by (*simp add*: *find-Some-set-membership*, *fastforce*)
let *?*Ψ' = *remove1* (*?*χ, *?*γ) Ψ
from *B* $\diamondsuit$(*2*) **have**
mset (*map snd* Δ) $\subseteq\#$ *mset* (*map* (*uncurry* ($\rightarrow$)) Ψ @ Γ $\ominus$ *map snd* Ψ) $-$ {# *?*χ $\rightarrow$ *?*γ #}
by (*simp add*: *insert-subset-eq-iff*)
moreover
have *mset* (*map* (*uncurry* ($\rightarrow$)) Ψ)
= *add-mset* (*case* (*fst* ψ, *snd* ψ) *of* (*x*, *xa*) $\Rightarrow$ *x* $\rightarrow$ *xa*)
(*image-mset* (*uncurry* ($\rightarrow$)) (*mset* (*remove1* (*fst* ψ, *snd* ψ) Ψ)))
by (*metis* (*no-types*)
A
image-mset-add-mset
in-multiset-in-set
insert-DiffM
mset-map
mset-remove1
uncurry-def)
ultimately have
mset (*map snd* Δ) $\subseteq\#$ *mset* (*map* (*uncurry* ($\rightarrow$)) *?*Ψ' @ Γ $\ominus$ *map snd* Ψ)
using
add-diff-cancel-left'
add-diff-cancel-right
diff-diff-add-mset
diff-subset-eq-self
mset-append
subset-eq-diff-conv
subset-mset.diff-add
by *auto*
moreover from *A B* $\diamondsuit$
have *mset* (Γ $\ominus$ *map snd* Ψ) = *mset*((*remove1 ?*γ Γ) $\ominus$ (*remove1 ?*γ (*map snd* Ψ)))
using
image-eqI
prod.sel(*2*)
set-map
by *force*
with *A* **have**

```
    mset (Γ ⊖ map snd Ψ) = mset((remove1 ?γ Γ) ⊖ (map snd ?Ψ'))
    by (metis
        remove1-pairs-list-projections-snd
        in-multiset-in-set
        list-subtract-mset-homomorphism
        mset-remove1)
  ultimately have
    mset (map snd Δ) ⊆# mset (map (uncurry (→)) ?Ψ'
                          @ (remove1 ?γ Γ)
                          ⊖ map snd ?Ψ')
    by simp
  hence mset (map snd (ℑ ?Ψ' Δ)) ⊆# mset (remove1 ?γ Γ)
    using Cons ◇(1) A
    by (metis (no-types, lifting)
          image-mset-add-mset
          in-multiset-in-set
          insert-DiffM
          insert-subset-eq-iff
          mset-map mset-remove1
          prod.collapse)
  with ◇(1) A have mset (map snd (ℑ ?Ψ' Δ)) + {# ?γ #} ⊆# mset Γ
    by (metis add-mset-add-single
          image-eqI
          insert-subset-eq-iff
          mset-remove1
          mset-subset-eqD
          set-map
          set-mset-mset
          snd-conv)
  hence mset (map snd ((fst δ ⊓ ?χ, ?γ) # (ℑ ?Ψ' Δ))) ⊆# mset Γ
    by simp
  moreover from ψ have
    ℑ Ψ (δ # Δ) = (fst δ ⊓ ?χ, ?γ) # (ℑ ?Ψ' Δ)
    by simp
  ultimately show ?thesis by simp
 qed
}
thus ?case by blast
qed
with assms show ?thesis by blast
qed

lemma (in classical-logic) right-merge-witness-stronger-theory:
 map (uncurry (⊔)) Δ ≼ map (uncurry (⊔)) (ℑ Ψ Δ)
proof −
 have ∀ Ψ. map (uncurry (⊔)) Δ ≼ map (uncurry (⊔)) (ℑ Ψ Δ)
 proof (induct Δ)
  case Nil
  then show ?case by simp
```

```
next
  case (Cons δ Δ)
  {
    fix Ψ
    have map (uncurry (⊔)) (δ # Δ) ≼ map (uncurry (⊔)) (𝔍 Ψ (δ # Δ))
    proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
      case True
      hence 𝔍 Ψ (δ # Δ) = δ # 𝔍 Ψ Δ
        by simp
      moreover have ⊢ (uncurry (⊔)) δ → (uncurry (⊔)) δ
        by (metis axiom-k axiom-s modus-ponens)
      ultimately show ?thesis using Cons
        by (simp add: stronger-theory-left-right-cons)
    next
      case False
      from this obtain ψ where ψ:
        find (λψ. uncurry (→) ψ = snd δ) Ψ = Some ψ
        by fastforce
      let ?χ = fst ψ
      let ?γ = snd ψ
      let ?μ = fst δ
      have uncurry (→) = (λ ψ. fst ψ → snd ψ)
           uncurry (⊔) = (λ δ. fst δ ⊔ snd δ)
        by fastforce+
      hence uncurry (⊔) δ = ?μ ⊔ (?χ → ?γ)
        using ψ find-Some-predicate
        by fastforce
      moreover
      {
        fix μ χ γ
        have ⊢ ((μ ⊓ χ) ⊔ γ) → (μ ⊔ (χ → γ))
        proof −
          have ∀ 𝔐. 𝔐 ⊨_prop ((⟨μ⟩ ⊓ ⟨χ⟩) ⊔ ⟨γ⟩) → (⟨μ⟩ ⊔ (⟨χ⟩ → ⟨γ⟩))
            by fastforce
          hence ⊢ ⦇ ((⟨μ⟩ ⊓ ⟨χ⟩) ⊔ ⟨γ⟩) → (⟨μ⟩ ⊔ (⟨χ⟩ → ⟨γ⟩)) ⦈
            using propositional-semantics by blast
          thus ?thesis
            by simp
        qed
      }
      ultimately show ?thesis
        using Cons ψ stronger-theory-left-right-cons
        by simp
    qed
  }
  thus ?case by blast
qed
thus ?thesis by blast
qed
```

```
lemma (in classical-logic) left-merge-witness-stronger-theory:
  map (uncurry (⊔)) Ψ ≼ map (uncurry (⊔)) (𝔍 Ψ Δ)
proof −
  have ∀ Ψ. map (uncurry (⊔)) Ψ ≼ map (uncurry (⊔)) (𝔍 Ψ Δ)
  proof (induct Δ)
    case Nil
    then show ?case
      by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have map (uncurry (⊔)) Ψ ≼ map (uncurry (⊔)) (𝔍 Ψ (δ # Δ))
      proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
        case True
        then show ?thesis
          using Cons stronger-theory-right-cons
          by auto
      next
        case False
        from this obtain ψ where ψ:
          find (λψ. uncurry (→) ψ = snd δ) Ψ = Some ψ
          by fastforce
        let ?χ = fst ψ
        let ?γ = snd ψ
        let ?μ = fst δ
        have uncurry (→) = (λ ψ. fst ψ → snd ψ)
             uncurry (⊔) = (λ δ. fst δ ⊔ snd δ)
          by fastforce+
        hence
          uncurry (⊔) δ = ?μ ⊔ (?χ → ?γ)
          uncurry (⊔) ψ = ?χ ⊔ ?γ
          using ψ find-Some-predicate
          by fastforce+
        moreover
        {
          fix μ χ γ
          have ⊢ ((μ ⊓ χ) ⊔ γ) → (χ ⊔ γ)
          proof −
            have ∀ 𝔐. 𝔐 ⊨_prop (((⟨μ⟩ ⊓ ⟨χ⟩) ⊔ ⟨γ⟩) → (⟨χ⟩ ⊔ ⟨γ⟩))
              by fastforce
            hence ⊢ (| (((⟨μ⟩ ⊓ ⟨χ⟩) ⊔ ⟨γ⟩) → (⟨χ⟩ ⊔ ⟨γ⟩)) |)
              using propositional-semantics by blast
            thus ?thesis
              by simp
          qed
        }
        ultimately have
```

```
        map (uncurry (⊔)) (ψ # (remove1 ψ Ψ)) ≼
         map (uncurry (⊔)) (𝔍 Ψ (δ # Δ))
        using Cons ψ stronger-theory-left-right-cons
        by simp
      moreover from ψ have ψ ∈ set Ψ
        by (simp add: find-Some-set-membership)
      hence mset (map (uncurry (⊔)) (ψ # (remove1 ψ Ψ))) =
            mset (map (uncurry (⊔)) Ψ)
        by (metis insert-DiffM
                  mset.simps(2)
                  mset-map
                  mset-remove1
                  set-mset-mset)
      hence map (uncurry (⊔)) Ψ ≼ map (uncurry (⊔)) (ψ # (remove1 ψ Ψ))
        by (simp add: msub-stronger-theory-intro)
      ultimately show ?thesis
        using stronger-theory-transitive by blast
    qed
  }
  then show ?case by blast
 qed
 thus ?thesis by blast
qed

lemma (in classical-logic) measure-empty-deduction:
 [] $⊢ Φ = (∀ φ ∈ set Φ. ⊢ φ)
 by (induct Φ, simp, rule iffI, fastforce+)

lemma (in classical-logic) measure-stronger-theory-left-monotonic:
 assumes Σ ≼ Γ
    and Σ $⊢ Φ
  shows Γ $⊢ Φ
 using assms
proof (induct Φ arbitrary: Σ Γ)
 case Nil
 then show ?case by simp
next
 case (Cons φ Φ)
 from this obtain Ψ Δ where
  Ψ: mset (map snd Ψ) ⊆# mset Σ
     map (uncurry (⊔)) Ψ :⊢ φ
     map (uncurry (→)) Ψ @ Σ ⊖ (map snd Ψ) $⊢ Φ
  and
  Δ: map snd Δ = Σ
     mset (map fst Δ) ⊆# mset Γ
     ∀ (γ,σ) ∈ set Δ. ⊢ γ → σ
  unfolding stronger-theory-relation-def
  by fastforce
 from ‹mset (map snd Ψ) ⊆# mset Σ›
```

```
    ‹map snd Δ = Σ›
  obtain Ω where Ω:
    map (λ (ψ, σ, -). (ψ, σ)) Ω = Ψ
    mset (map (λ (-, σ, γ). (γ, σ)) Ω) ⊆# mset Δ
    using triple-list-exists by blast
  let ?Θ = map (λ (ψ, -, γ). (ψ, γ)) Ω
  have map snd ?Θ = map fst (map (λ (-, σ, γ). (γ, σ)) Ω)
    by auto
  hence mset (map snd ?Θ) ⊆# mset Γ
    using Ω(2) Δ(2) map-monotonic subset-mset.order-trans
    by metis
  moreover have map (uncurry (⊔)) Ψ ≼ map (uncurry (⊔)) ?Θ
  proof −
    let ?Φ = map (λ (ψ, σ, γ). (ψ ⊔ γ, ψ ⊔ σ)) Ω
    have map snd ?Φ = map (uncurry (⊔)) Ψ
      using Ω(1) by fastforce
    moreover have map fst ?Φ = map (uncurry (⊔)) ?Θ
      by fastforce
    hence mset (map fst ?Φ) ⊆# mset (map (uncurry (⊔)) ?Θ)
      by (metis subset-mset.dual-order.refl)
    moreover
    have mset (map (λ(ψ, σ, -). (ψ, σ)) Ω) ⊆# mset Ψ
      using Ω(1) by simp
    hence ∀ (φ,χ) ∈ set ?Φ. ⊢ φ → χ using Ω(2)
    proof (induct Ω)
      case Nil
      then show ?case by simp
    next
      case (Cons ω Ω)
      let ?Φ = map (λ (ψ, σ, γ). (ψ ⊔ γ, ψ ⊔ σ)) (ω # Ω)
      let ?Φ′ = map (λ (ψ, σ, γ). (ψ ⊔ γ, ψ ⊔ σ)) Ω
      have mset (map (λ(ψ, σ, -). (ψ, σ)) Ω) ⊆# mset Ψ
           mset (map (λ(-, σ, γ). (γ, σ)) Ω) ⊆# mset Δ
        using Cons.prems(1) Cons.prems(2) subset-mset.dual-order.trans by fast-
force+
      with Cons have ∀ (φ,χ) ∈ set ?Φ′. ⊢ φ → χ by fastforce
      moreover
      let ?ψ = (λ (ψ, -, -). ψ) ω
      let ?σ = (λ (-, σ, -). σ) ω
      let ?γ = (λ (-, -, γ). γ) ω
      have (λ(-, σ, γ). (γ, σ)) = (λ ω. ((λ (-, -, γ). γ) ω,(λ (-, σ, -). σ) ω)) by auto
      hence (λ(-, σ, γ). (γ, σ)) ω = (?γ, ?σ) by metis
      hence ⊢ ?γ → ?σ
        using Cons.prems(2) mset-subset-eqD Δ(3)
        by fastforce
      hence ⊢ (?ψ ⊔ ?γ) → (?ψ ⊔ ?σ)
        unfolding disjunction-def
        using modus-ponens hypothetical-syllogism
        by blast
```

```
    moreover have
      (λ(ψ, σ, γ). (ψ ⊔ γ, ψ ⊔ σ)) =
       (λ ω. (((λ (ψ, -, -). ψ) ω) ⊔ ((λ (-, -, γ). γ) ω),
              ((λ (ψ, -, -). ψ) ω) ⊔ ((λ (-, σ, -). σ) ω)))
      by auto
    hence (λ(ψ, σ, γ). (ψ ⊔ γ, ψ ⊔ σ)) ω = ((?ψ ⊔ ?γ), (?ψ ⊔ ?σ)) by metis
    ultimately show ?case by simp
  qed
  ultimately show ?thesis
    unfolding stronger-theory-relation-def
    by blast
qed
hence map (uncurry (⊔)) ?Θ :⊢ φ
  using Ψ(2)
        stronger-theory-deduction-monotonic
          [where Σ=map (uncurry (⊔)) Ψ
            and Γ=map (uncurry (⊔)) ?Θ
            and φ=φ]
  by metis
moreover have
  (map (uncurry (→)) Ψ @ Σ ⊖ (map snd Ψ)) ≼
   (map (uncurry (→)) ?Θ @ Γ ⊖ (map snd ?Θ))
proof −
  have map (uncurry (→)) Ψ ≼ map (uncurry (→)) ?Θ
  proof −
    let ?Φ = map (λ (ψ, σ, γ). (ψ → γ, ψ → σ)) Ω
    have map snd ?Φ = map (uncurry (→)) Ψ
      using Ω(1) by fastforce
    moreover have map fst ?Φ = map (uncurry (→)) ?Θ
      by fastforce
    hence mset (map fst ?Φ) ⊆# mset (map (uncurry (→)) ?Θ)
      by (metis subset-mset.dual-order.refl)
    moreover
    have mset (map (λ(ψ, σ, -). (ψ, σ)) Ω) ⊆# mset Ψ
      using Ω(1) by simp
    hence ∀ (φ,χ) ∈ set ?Φ. ⊢ φ → χ using Ω(2)
    proof (induct Ω)
      case Nil
      then show ?case by simp
    next
      case (Cons ω Ω)
      let ?Φ = map (λ (ψ, σ, γ). (ψ → γ, ψ → σ)) (ω # Ω)
      let ?Φ' = map (λ (ψ, σ, γ). (ψ → γ, ψ → σ)) Ω
      have mset (map (λ(ψ, σ, -). (ψ, σ)) Ω) ⊆# mset Ψ
           mset (map (λ(-, σ, γ). (γ, σ)) Ω) ⊆# mset Δ
        using Cons.prems(1) Cons.prems(2) subset-mset.dual-order.trans by
fastforce+
      with Cons have ∀ (φ,χ) ∈ set ?Φ'. ⊢ φ → χ by fastforce
      moreover
```

let *?ψ* = (λ (*ψ*, -, -). *ψ*) *ω*
let *?σ* = (λ (-, *σ*, -). *σ*) *ω*
let *?γ* = (λ (-, -, *γ*). *γ*) *ω*
have (λ(-, *σ*, *γ*). (*γ*, *σ*)) = (λ *ω*. ((λ (-, -, *γ*). *γ*) *ω*,(λ (-, *σ*, -). *σ*) *ω*)) **by** *auto*
hence (λ(-, *σ*, *γ*). (*γ*, *σ*)) *ω* = (*?γ*, *?σ*) **by** *metis*
hence ⊢ *?γ* → *?σ*
using *Cons.prems*(*2*) *mset-subset-eqD* Δ(*3*)
by *fastforce*
hence ⊢ (*?ψ* → *?γ*) → (*?ψ* → *?σ*)
using *modus-ponens hypothetical-syllogism*
by *blast*
moreover have
(λ(*ψ*, *σ*, *γ*). (*ψ* → *γ*, *ψ* → *σ*)) =
(λ *ω*. (((λ (*ψ*, -, -). *ψ*) *ω*) → ((λ (-, -, *γ*). *γ*) *ω*),
((λ (*ψ*, -, -). *ψ*) *ω*) → ((λ (-, *σ*, -). *σ*) *ω*)))
by *auto*
hence (λ(*ψ*, *σ*, *γ*). (*ψ* → *γ*, *ψ* → *σ*)) *ω* = ((*?ψ* → *?γ*), (*?ψ* → *?σ*)) **by** *metis*
ultimately show *?case* **by** *simp*
qed
ultimately show *?thesis*
unfolding *stronger-theory-relation-def*
by *blast*
qed
moreover
have (Σ ⊖ (*map snd* Ψ)) ≼ (Γ ⊖ (*map snd ?*Θ))
proof −
let *?*Δ = Δ ⊖ (*map* (λ (-, *σ*, *γ*). (*γ*, *σ*)) Ω)
have *mset* (*map fst ?*Δ) ⊆# *mset* (Γ ⊖ (*map snd ?*Θ))
using Δ(*2*)
by (*metis* Ω(*2*)
‹*map snd* (*map* (λ(*ψ*, -, *γ*). (*ψ*, *γ*)) Ω) =
map fst (*map* (λ(-, *σ*, *γ*). (*γ*, *σ*)) Ω)›
list-subtract-monotonic
map-list-subtract-mset-equivalence)
moreover
from Ω(*2*) **have** *mset ?*Δ ⊆# *mset* Δ **by** *simp*
hence ∀ (*γ*,*σ*) ∈ *set ?*Δ. ⊢ *γ* → *σ*
using Δ(*3*)
by (*metis mset-subset-eqD set-mset-mset*)
moreover
have *map snd* (*map* (λ(-, *σ*, *γ*). (*γ*, *σ*)) Ω) = *map snd* Ψ
using Ω(*1*)
by (*induct* Ω, *simp*, *fastforce*)
hence *mset* (*map snd ?*Δ) = *mset* (Σ ⊖ (*map snd* Ψ))
by (*metis* Δ(*1*) Ω(*2*) *map-list-subtract-mset-equivalence*)
ultimately show *?thesis*
by (*metis stronger-theory-relation-alt-def*)
qed

ultimately show *?thesis* **using** *stronger-theory-combine* **by** *blast*
qed
hence *map* (*uncurry* (→)) *?Θ* @ Γ ⊖ (*map snd ?Θ*) $⊢ Φ
using Ψ(*3*) *Cons* **by** *blast*
ultimately show *?case*
by (*metis measure-deduction.simps*(*2*))
qed

lemma (**in** *classical-logic*) *merge-witness-measure-deduction-intro*:
assumes *mset* (*map snd* Δ) ⊆# *mset* (*map* (*uncurry* (→)) Ψ @ Γ ⊖ (*map snd* Ψ))
and *map* (*uncurry* (→)) Δ @ (*map* (*uncurry* (→)) Ψ @ Γ ⊖ *map snd* Ψ) ⊖ *map snd* Δ $⊢ Φ
(**is** *?Γ*$_0$ $⊢ Φ)
shows *map* (*uncurry* (→)) (ℑ Ψ Δ) @ Γ ⊖ *map snd* (ℑ Ψ Δ) $⊢ Φ
(**is** *?Γ* $⊢ Φ)
proof −
let *?Σ* = 𝔅 Ψ Δ
let *?A* = *map* (*uncurry* (→)) Δ
let *?B* = *map* (*uncurry* (→)) Ψ
let *?C* = *map snd ?Σ*
let *?D* = Γ ⊖ (*map snd* Ψ)
let *?E* = *map snd* (Δ ⊖ *?Σ*)
have Σ: *mset ?Σ* ⊆# *mset* Δ
mset ?C ⊆# *mset ?B*
mset ?E ⊆# *mset ?D*
using *assms*(*1*)
second-component-msub
second-component-snd-projection-msub
second-component-diff-msub
by *simp*+
moreover
from *calculation* **have**
image-mset snd (*mset* Δ − *mset* (𝔅 Ψ Δ))
⊆# *mset* Γ − *image-mset snd* (*mset* Ψ)
by *simp*
hence *mset* Γ − *image-mset snd* (*mset* Ψ)
− *image-mset snd* (*mset* Δ − *mset* (𝔅 Ψ Δ))
+ *image-mset snd* (*mset* Δ − *mset* (𝔅 Ψ Δ))
= *mset* Γ − *image-mset snd* (*mset* Ψ)
using *subset-mset.diff-add* **by** *blast*
then have *image-mset snd* (*mset* Δ − *mset* (𝔅 Ψ Δ))
+ ({#*x* → *y*. (*x*, *y*) ∈# *mset* Ψ#}
+ (*mset* Γ − (*image-mset snd* (*mset* Ψ)
+ *image-mset snd* (*mset* Δ − *mset* (𝔅 Ψ Δ)))))
= {#*x* → *y*. (*x*, *y*) ∈# *mset* Ψ#} + (*mset* Γ − *image-mset snd* (*mset* Ψ))
by (*simp add*: *union-commute*)
with *calculation* **have** *mset ?Γ*$_0$ = *mset* (*?A* @ (*?B* ⊖ *?C*) @ (*?D* ⊖ *?E*))
by (*simp*, *metis* (*no-types*) *add-diff-cancel-left image-mset-union subset-mset.diff-add*)

```
  moreover have (?A @ (?B ⊖ ?C)) ⪯ map (uncurry (→)) (𝔍 Ψ Δ)
    using second-component-merge-witness-stronger-theory by simp
  moreover have mset (?D ⊖ ?E) = mset (Γ ⊖ map snd (𝔍 Ψ Δ))
    using second-component-merge-witness-snd-projection
    by simp
  with calculation have (?A @ (?B ⊖ ?C) @ (?D ⊖ ?E)) ⪯ ?Γ
    by (metis
        (no-types, lifting)
        stronger-theory-combine
        append.assoc
        list-subtract-mset-homomorphism
        msub-stronger-theory-intro
        map-list-subtract-mset-containment
        map-list-subtract-mset-equivalence
        mset-subset-eq-add-right
        subset-mset.add-diff-inverse
        subset-mset.diff-add-assoc2)
  ultimately have ?Γ₀ ⪯ ?Γ
    unfolding stronger-theory-relation-alt-def
    by simp
  thus ?thesis
    using assms(2) measure-stronger-theory-left-monotonic
    by blast
qed

lemma (in classical-logic) measure-formula-right-split:
  Γ $⊢ (ψ ⊔ φ # ψ → φ # Φ) = Γ $⊢ (φ # Φ)
proof (rule iffI)
  assume Γ $⊢ (φ # Φ)
  from this obtain Ψ where Ψ:
    mset (map snd Ψ) ⊆# mset Γ
    map (uncurry (⊔)) Ψ :⊢ φ
    (map (uncurry (→)) Ψ @ Γ ⊖ (map snd Ψ)) $⊢ Φ
    by auto
  let ?Ψ₁ = zip (map (λ (χ,γ). ψ ⊔ χ) Ψ) (map snd Ψ)
  let ?Γ₁ = map (uncurry (→)) ?Ψ₁ @ Γ ⊖ (map snd ?Ψ₁)
  let ?Ψ₂ = zip (map (λ (χ,γ). ψ → χ) Ψ) (map (uncurry (→)) ?Ψ₁)
  let ?Γ₂ = map (uncurry (→)) ?Ψ₂ @ ?Γ₁ ⊖ (map snd ?Ψ₂)
  have map (uncurry (→)) Ψ ⪯ map (uncurry (→)) ?Ψ₂
  proof (induct Ψ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Ψ)
    let ?χ = fst δ
    let ?γ = snd δ
    let ?Ψ₁ = zip (map (λ (χ,γ). ψ ⊔ χ) Ψ) (map snd Ψ)
    let ?Ψ₂ = zip (map (λ (χ,γ). ψ → χ) Ψ) (map (uncurry (→)) ?Ψ₁)
    let ?T₁ = λ Ψ. map (uncurry (→)) (zip (map (λ (χ,γ). ψ ⊔ χ) Ψ) (map snd
```

Ψ))
  let *?T*$_2$ = λ Ψ. *map* (*uncurry* ($\rightarrow$)) (*zip* (*map* (λ (χ,γ). $\psi \rightarrow \chi$) Ψ) (*?T*$_1$ Ψ))
  {
    fix δ :: *'a* × *'a*
    have (λ (χ,γ). $\psi \sqcup \chi$) = (λ δ. $\psi \sqcup$ (*fst* δ))
        (λ (χ,γ). $\psi \rightarrow \chi$) = (λ δ. $\psi \rightarrow$ (*fst* δ))
      by *fastforce*+
    note *functional-identities* = *this*
    have (λ (χ,γ). $\psi \sqcup \chi$) δ = $\psi \sqcup$ (*fst* δ)
        (λ (χ,γ). $\psi \rightarrow \chi$) δ = $\psi \rightarrow$ (*fst* δ)
      by (*simp add*: *functional-identities*)+
  }
  hence *?T*$_2$ (δ # Ψ) = (($\psi \rightarrow$ *?*χ) $\rightarrow$ ($\psi \sqcup$ *?*χ) $\rightarrow$ *?*γ) # (*map* (*uncurry* ($\rightarrow$)) *?*Ψ_2)
    by *simp*
  moreover have *map* (*uncurry* ($\rightarrow$)) (δ # Ψ) = (*?*$\chi \rightarrow$ *?*γ) # *map* (*uncurry* ($\rightarrow$)) Ψ
    by (*simp add*: *case-prod-beta*)
  moreover
  {
    fix χ ψ γ
    have $\vdash$ (($\psi \rightarrow \chi$) $\rightarrow$ ($\psi \sqcup \chi$) $\rightarrow \gamma$) $\leftrightarrow$ ($\chi \rightarrow \gamma$)
    proof −
      have $\forall$ $\mathfrak{M}$. $\mathfrak{M} \models_{prop}$ ((⟨ψ⟩ $\rightarrow$ ⟨χ⟩) $\rightarrow$ (⟨ψ⟩ $\sqcup$ ⟨χ⟩) $\rightarrow$ ⟨γ⟩) $\leftrightarrow$ (⟨χ⟩ $\rightarrow$ ⟨γ⟩)
        by *fastforce*
      hence $\vdash$ ⦇ ((⟨ψ⟩ $\rightarrow$ ⟨χ⟩) $\rightarrow$ (⟨ψ⟩ $\sqcup$ ⟨χ⟩) $\rightarrow$ ⟨γ⟩) $\leftrightarrow$ (⟨χ⟩ $\rightarrow$ ⟨γ⟩) ⦈
        using *propositional-semantics* **by** *blast*
      thus *?thesis* **by** *simp*
    qed
  }
  hence *identity*: $\vdash$ (($\psi \rightarrow$ *?*χ) $\rightarrow$ ($\psi \sqcup$ *?*χ) $\rightarrow$ *?*γ) $\rightarrow$ (*?*$\chi \rightarrow$ *?*γ)
    using *biconditional-def* **by** *auto*
  assume *map* (*uncurry* ($\rightarrow$)) $\Psi \preceq$ *map* (*uncurry* ($\rightarrow$)) *?*Ψ_2
  with *identity* **have** ((*?*$\chi \rightarrow$ *?*γ) # *map* (*uncurry* ($\rightarrow$)) Ψ) $\preceq$
                (((*?*$\psi \rightarrow$ *?*χ) $\rightarrow$ ($\psi \sqcup$ *?*χ) $\rightarrow$ *?*γ) # (*map* (*uncurry* ($\rightarrow$)) *?*Ψ_2))
    using *stronger-theory-left-right-cons* **by** *blast*
  ultimately show *?case* **by** *simp*
qed
hence (*map* (*uncurry* ($\rightarrow$)) Ψ @ Γ ⊖ (*map snd* Ψ)) $\preceq$
      ((*map* (*uncurry* ($\rightarrow$)) *?*Ψ_2) @ Γ ⊖ (*map snd* Ψ))
  using *stronger-theory-combine stronger-theory-reflexive* **by** *blast*
moreover have *mset* *?*Γ_2 = *mset* ((*map* (*uncurry* ($\rightarrow$)) *?*Ψ_2) @ Γ ⊖ (*map snd* *?*Ψ_1))
  by *simp*
ultimately have (*map* (*uncurry* ($\rightarrow$)) Ψ @ Γ ⊖ (*map snd* Ψ)) $\preceq$ *?*Γ_2
  by (*simp add*: *stronger-theory-relation-def*)
hence *?*Γ_2 $\$\vdash$ Φ
  using Ψ(*3*) *measure-stronger-theory-left-monotonic* **by** *blast*
moreover

have (*map* (*uncurry* (⊔)) *?*Ψ$_2$) :⊢ $\psi \to \varphi$
proof −
let *?*Γ = *map* (λ (χ, γ). (ψ → χ) ⊔ (ψ ⊔ χ) → γ) Ψ
let *?*Σ = *map* (λ (χ, γ). (ψ → (χ ⊔ γ))) Ψ
have *map* (*uncurry* (⊔)) *?*Ψ$_2$ = *?*Γ
proof (*induct* Ψ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* χ Ψ)
have (λ φ. (*case* φ *of* (χ, γ) ⇒ ψ → χ) ⊔ (*case* φ *of* (χ, γ) ⇒ ψ ⊔ χ) → *snd* φ) =
(λ φ. (*case* φ *of* (χ, γ) ⇒ ψ → χ ⊔ (ψ ⊔ χ) → γ))
by *fastforce*
hence (*case* χ *of* (χ, γ) ⇒ ψ → χ) ⊔ (*case* χ *of* (χ, γ) ⇒ ψ ⊔ χ) → *snd* χ =
(*case* χ *of* (χ, γ) ⇒ ψ → χ ⊔ (ψ ⊔ χ) → γ)
by *metis*
with *Cons* **show** *?case* **by** *simp*
qed
moreover have *?*Σ ≼ *?*Γ
proof (*induct* Ψ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* δ Ψ)
let *?*α = (λ (χ, γ). (ψ → χ) ⊔ (ψ ⊔ χ) → γ) δ
let *?*β = (λ (χ, γ). (ψ → (χ ⊔ γ))) δ
let *?*χ = *fst* δ
let *?*γ = *snd* δ
have (λ δ. (*case* δ *of* (χ, γ) ⇒ ψ → χ ⊔ (ψ ⊔ χ) → γ)) =
(λ δ. ψ → *fst* δ ⊔ (ψ ⊔ *fst* δ) → *snd* δ)
(λ δ. (*case* δ *of* (χ, γ) ⇒ ψ → (χ ⊔ γ))) = (λ δ. ψ → (*fst* δ ⊔ *snd* δ))
by *fastforce*+
hence *?*α = (ψ → *?*χ) ⊔ (ψ ⊔ *?*χ) → *?*γ
*?*β = ψ → (*?*χ ⊔ *?*γ)
by *metis*+
moreover
{
fix ψ χ γ
have ⊢ ((ψ → χ) ⊔ (ψ ⊔ χ) → γ) → (ψ → (χ ⊔ γ))
proof −
have ∀ 𝔐. 𝔐 ⊨$_{prop}$ ((⟨ψ⟩ → ⟨χ⟩) ⊔ (⟨ψ⟩ ⊔ ⟨χ⟩) → ⟨γ⟩) → (⟨ψ⟩ → (⟨χ⟩ ⊔ ⟨γ⟩))
by *fastforce*
hence ⊢ ⦇ ((⟨ψ⟩ → ⟨χ⟩) ⊔ (⟨ψ⟩ ⊔ ⟨χ⟩) → ⟨γ⟩) → (⟨ψ⟩ → (⟨χ⟩ ⊔ ⟨γ⟩)) ⦈
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
}

```
  ultimately have ⊢ ?α → ?β by simp
  thus ?case
    using Cons
          stronger-theory-left-right-cons
    by simp
qed
moreover have ∀ φ. (map (uncurry (⊔)) Ψ) :⊢ φ ⟶ ?Σ :⊢ ψ → φ
proof (induct Ψ)
  case Nil
  then show ?case
    using axiom-k modus-ponens
    by fastforce
next
  case (Cons δ Ψ)
  let ?δ' = (λ (χ, γ). (ψ → (χ ⊔ γ))) δ
  let ?Σ = map (λ (χ, γ). (ψ → (χ ⊔ γ))) Ψ
  let ?Σ' = map (λ (χ, γ). (ψ → (χ ⊔ γ))) (δ # Ψ)
  {
    fix φ
    assume map (uncurry (⊔)) (δ # Ψ) :⊢ φ
    hence map (uncurry (⊔)) Ψ :⊢ (uncurry (⊔)) δ → φ
      using list-deduction-theorem
      by simp
    hence ?Σ :⊢ ψ → (uncurry (⊔)) δ → φ
      using Cons
      by blast
    moreover
    {
      fix α β γ
      have ⊢ (α → β → γ) → ((α → β) → α → γ)
        using axiom-s by auto
    }
    ultimately have ?Σ :⊢ (ψ → (uncurry (⊔)) δ) → ψ → φ
      using list-deduction-weaken [where ?Γ=?Σ]
            list-deduction-modus-ponens [where ?Γ=?Σ]
      by metis
    moreover
    have (λ δ. ψ → (uncurry (⊔)) δ) = (λ δ. (λ (χ, γ). (ψ → (χ ⊔ γ))) δ)
      by fastforce
    ultimately have ?Σ :⊢ (λ (χ, γ). (ψ → (χ ⊔ γ))) δ → ψ → φ
      by metis
    hence ?Σ' :⊢ ψ → φ
      using list-deduction-theorem
      by simp
  }
  then show ?case by simp
qed
with Ψ(2) have ?Σ :⊢ ψ → φ
  by blast
```

ultimately show *?thesis*
using *stronger-theory-deduction-monotonic* **by** *auto*
qed
moreover have *mset* (*map snd* $?\Psi_2$) $\subseteq\#$ *mset* $?\Gamma_1$ **by** *simp*
ultimately have $?\Gamma_1$ $\$\vdash$ $(\psi \to \varphi \# \Phi)$ **using** *measure-deduction.simps*(*2*) **by** *blast*
moreover have $\vdash$ (*map* (*uncurry* ($\sqcup$)) Ψ :→ φ) → (*map* (*uncurry* ($\sqcup$)) $?\Psi_1$) :→ ($\psi \sqcup \varphi$)
proof (*induct* Ψ)
case *Nil*
then show *?case*
unfolding *disjunction-def*
using *axiom-k modus-ponens*
by *fastforce*
next
case (*Cons* ν Ψ)
let $?\Delta$ = *map* (*uncurry* ($\sqcup$)) Ψ
let $?\Delta'$ = *map* (*uncurry* ($\sqcup$)) (ν # Ψ)
let $?\Sigma$ = *map* (*uncurry* ($\sqcup$)) (*zip* (*map* (λ (χ,γ). $\psi \sqcup \chi$) Ψ) (*map snd* Ψ))
let $?\Sigma'$ = *map* (*uncurry* ($\sqcup$)) (*zip* (*map* (λ (χ,γ). $\psi \sqcup \chi$) (ν # Ψ)) (*map snd* (ν # Ψ)))
have $\vdash$ ($?\Delta'$:→ φ) → (*uncurry* ($\sqcup$)) ν → $?\Delta$:→ φ
by (*simp*, *metis axiom-k axiom-s modus-ponens*)
with *Cons* **have** $\vdash$ ($?\Delta'$:→ φ) → (*uncurry* ($\sqcup$)) ν → $?\Sigma$:→ ($\psi \sqcup \varphi$)
using *hypothetical-syllogism modus-ponens*
by *blast*
hence ($?\Delta'$:→ φ) # ((*uncurry* ($\sqcup$)) ν) # $?\Sigma$:⊢ $\psi \sqcup \varphi$
by (*simp add*: *list-deduction-def*)
moreover have *set* (($?\Delta'$:→ φ) # ((*uncurry* ($\sqcup$)) ν) # $?\Sigma$) =
set (((*uncurry* ($\sqcup$)) ν) # ($?\Delta'$:→ φ) # $?\Sigma$)
by *fastforce*
ultimately have ((*uncurry* ($\sqcup$)) ν) # ($?\Delta'$:→ φ) # $?\Sigma$:⊢ $\psi \sqcup \varphi$
using *list-deduction-monotonic* **by** *blast*
hence ($?\Delta'$:→ φ) # $?\Sigma$:⊢ ((*uncurry* ($\sqcup$)) ν) → ($\psi \sqcup \varphi$)
using *list-deduction-theorem*
by *simp*
moreover
let $?\chi$ = *fst* ν
let $?\gamma$ = *snd* ν
have (λ ν . (*uncurry* ($\sqcup$)) ν) = (λ ν. *fst* $\nu \sqcup$ *snd* ν)
by *fastforce*
hence (*uncurry* ($\sqcup$)) ν = $?\chi \sqcup ?\gamma$ **by** *simp*
ultimately have ($?\Delta'$:→ φ) # $?\Sigma$:⊢ ($?\chi \sqcup ?\gamma$) → ($\psi \sqcup \varphi$) **by** *simp*
moreover
{
fix α β δ γ
have $\vdash$ (($\beta \sqcup \alpha$) → ($\gamma \sqcup \delta$)) → (($\gamma \sqcup \beta$) $\sqcup \alpha$) → ($\gamma \sqcup \delta$)
proof −
have $\forall$ $\mathfrak{M}$. $\mathfrak{M}$ $\models_{prop}$ ((⟨β⟩ $\sqcup$ ⟨α⟩) → (⟨γ⟩ $\sqcup$ ⟨δ⟩)) → ((⟨γ⟩ $\sqcup$ ⟨β⟩) $\sqcup$ ⟨α⟩)

```
→ (⟨γ⟩ ⊔ ⟨δ⟩)
        by fastforce
      hence ⊢ ⦇ ((⟨β⟩ ⊔ ⟨α⟩) → (⟨γ⟩ ⊔ ⟨δ⟩)) → (((⟨γ⟩ ⊔ ⟨β⟩) ⊔ ⟨α⟩) → (⟨γ⟩ ⊔
⟨δ⟩)) ⦈
        using propositional-semantics by blast
      thus ?thesis by simp
    qed
  }
  hence (?Δ' :→ φ) # ?Σ :⊢ ((?χ ⊔ ?γ) → (ψ ⊔ φ)) → ((ψ ⊔ ?χ) ⊔ ?γ) → (ψ
⊔ φ)
    using list-deduction-weaken by blast
  ultimately have (?Δ' :→ φ) # ?Σ :⊢ ((ψ ⊔ ?χ) ⊔ ?γ) → (ψ ⊔ φ)
    using list-deduction-modus-ponens by blast
  hence ((ψ ⊔ ?χ) ⊔ ?γ) # (?Δ' :→ φ) # ?Σ :⊢ ψ ⊔ φ
    using list-deduction-theorem
    by simp
  moreover have set (((ψ ⊔ ?χ) ⊔ ?γ) # (?Δ' :→ φ) # ?Σ) =
                 set ((?Δ' :→ φ) # ((ψ ⊔ ?χ) ⊔ ?γ) # ?Σ)
    by fastforce
  moreover have
    map (uncurry (⊔)) (ν # Ψ) :→ φ
     # (ψ ⊔ fst ν) ⊔ snd ν
      # map (uncurry (⊔)) (zip (map (λ(-, a). ψ ⊔ a) Ψ) (map snd Ψ)) :⊢ (ψ ⊔
fst ν) ⊔ snd ν
    by (meson list.set-intros(1)
              list-deduction-monotonic
              list-deduction-reflection
              set-subset-Cons)
  ultimately have (?Δ' :→ φ) # ((ψ ⊔ ?χ) ⊔ ?γ) # ?Σ :⊢ ψ ⊔ φ
    using list-deduction-modus-ponens list-deduction-monotonic by blast
  moreover
  have (λ ν. ψ ⊔ fst ν) = (λ (χ, γ). ψ ⊔ χ)
    by fastforce
  hence ψ ⊔ fst ν = (λ (χ, γ). ψ ⊔ χ) ν
    by metis
  hence ((ψ ⊔ ?χ) ⊔ ?γ) # ?Σ = ?Σ'
    by simp
  ultimately have (?Δ' :→ φ) # ?Σ' :⊢ ψ ⊔ φ by simp
  then show ?case by (simp add: list-deduction-def)
 qed
 with Ψ(2) have map (uncurry (⊔)) ?Ψ1 :⊢ (ψ ⊔ φ)
  unfolding list-deduction-def
  using modus-ponens
  by blast
 moreover have mset (map snd ?Ψ1) ⊆# mset Γ using Ψ(1) by simp
 ultimately show Γ $⊢ (ψ ⊔ φ # ψ → φ # Φ)
  using measure-deduction.simps(2) by blast
next
 assume Γ $⊢ (ψ ⊔ φ # ψ → φ # Φ)
```

```
  from this obtain Ψ where Ψ:
    mset (map snd Ψ) ⊆# mset Γ
    map (uncurry (⊔)) Ψ :⊢ ψ ⊔ φ
    map (uncurry (→)) Ψ @ Γ ⊖ (map snd Ψ) $⊢ (ψ → φ # Φ)
    using measure-deduction.simps(2) by blast
  let ?Γ' = map (uncurry (→)) Ψ @ Γ ⊖ (map snd Ψ)
  from Ψ obtain Δ where Δ:
    mset (map snd Δ) ⊆# mset ?Γ'
    map (uncurry (⊔)) Δ :⊢ ψ → φ
    (map (uncurry (→)) Δ @ ?Γ' ⊖ (map snd Δ)) $⊢ Φ
    using measure-deduction.simps(2) by blast
  let ?Ω = ℑ Ψ Δ
  have mset (map snd ?Ω) ⊆# mset Γ
    using Δ(1) Ψ(1) merge-witness-msub-intro
    by blast
  moreover have map (uncurry (⊔)) ?Ω :⊢ φ
  proof −
    have map (uncurry (⊔)) ?Ω :⊢ ψ ⊔ φ
         map (uncurry (⊔)) ?Ω :⊢ ψ → φ
      using Ψ(2) Δ(2)
            stronger-theory-deduction-monotonic
            right-merge-witness-stronger-theory
            left-merge-witness-stronger-theory
      by blast+
    moreover
    have ⊢ (ψ ⊔ φ) → (ψ → φ) → φ
      unfolding disjunction-def
      using modus-ponens excluded-middle-elimination flip-implication
      by blast
    ultimately show ?thesis
      using list-deduction-weaken list-deduction-modus-ponens
      by blast
  qed
  moreover have map (uncurry (→)) ?Ω @ Γ ⊖ (map snd ?Ω) $⊢ Φ
    using Δ(1) Δ(3) Ψ(1) merge-witness-measure-deduction-intro by blast
  ultimately show Γ $⊢ (φ # Φ)
    using measure-deduction.simps(2) by blast
qed

primrec (in implication-logic)
  X-witness :: ('a × 'a) list ⇒ ('a × 'a) list ⇒ ('a × 'a) list (𝔛)
  where
    𝔛 Ψ [] = []
  | 𝔛 Ψ (δ # Δ) =
      (case find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ of
           None ⇒ δ # 𝔛 Ψ Δ
         | Some ψ ⇒ (fst ψ → fst δ, snd ψ) # (𝔛 (remove1 ψ Ψ) Δ))

primrec (in implication-logic)
```

```
  X-component :: ('a × 'a) list ⇒ ('a × 'a) list ⇒ ('a × 'a) list (𝔛•)
  where
    𝔛• Ψ [] = []
  | 𝔛• Ψ (δ # Δ) =
      (case find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ of
          None ⇒ 𝔛• Ψ Δ
        | Some ψ ⇒ (fst ψ → fst δ, snd ψ) # (𝔛• (remove1 ψ Ψ) Δ))

primrec (in implication-logic)
  Y-witness :: ('a × 'a) list ⇒ ('a × 'a) list ⇒ ('a × 'a) list (𝔜)
  where
    𝔜 Ψ [] = Ψ
  | 𝔜 Ψ (δ # Δ) =
      (case find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ of
          None ⇒ 𝔜 Ψ Δ
        | Some ψ ⇒ (fst ψ, (fst ψ → fst δ) → snd ψ) #
                   (𝔜 (remove1 ψ Ψ) Δ))

primrec (in implication-logic)
  Y-component :: ('a × 'a) list ⇒ ('a × 'a) list ⇒ ('a × 'a) list (𝔜•)
  where
    𝔜• Ψ [] = []
  | 𝔜• Ψ (δ # Δ) =
      (case find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ of
          None ⇒ 𝔜• Ψ Δ
        | Some ψ ⇒ (fst ψ, (fst ψ → fst δ) → snd ψ) #
                   (𝔜• (remove1 ψ Ψ) Δ))

lemma (in implication-logic) X-witness-right-empty [simp]:
  𝔛 [] Δ = Δ
  by (induct Δ, simp+)

lemma (in implication-logic) Y-witness-right-empty [simp]:
  𝔜 [] Δ = []
  by (induct Δ, simp+)

lemma (in implication-logic) X-witness-map-snd-decomposition:
   mset (map snd (𝔛 Ψ Δ)) = mset (map snd ((𝔄 Ψ Δ) @ (Δ ⊖ (𝔅 Ψ Δ))))
proof −
  have ∀ Ψ. mset (map snd (𝔛 Ψ Δ)) = mset (map snd ((𝔄 Ψ Δ) @ (Δ ⊖ (𝔅 Ψ
Δ))))
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have mset (map snd (𝔛 Ψ (δ # Δ)))
```

```
      = mset (map snd (𝔄 Ψ (δ # Δ) @ (δ # Δ) ⊖ 𝔅 Ψ (δ # Δ)))
    using Cons
    by (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None,
        simp,
        metis (no-types, lifting)
              add-mset-add-single
              image-mset-single
              image-mset-union
              mset-subset-eq-multiset-union-diff-commute
              second-component-msub,
        fastforce)
  }
  then show ?case by blast
 qed
 thus ?thesis by blast
qed

lemma (in implication-logic) Y-witness-map-snd-decomposition:
  mset (map snd (𝔜 Ψ Δ)) = mset (map snd ((Ψ ⊖ (𝔄 Ψ Δ)) @ (𝔜• Ψ Δ)))
proof −
 have ∀ Ψ. mset (map snd (𝔜 Ψ Δ)) = mset (map snd ((Ψ ⊖ (𝔄 Ψ Δ)) @ (𝔜•
Ψ Δ)))
 proof (induct Δ)
  case Nil
  then show ?case by simp
 next
  case (Cons δ Δ)
  {
   fix Ψ
   have mset (map snd (𝔜 Ψ (δ # Δ))) = mset (map snd (Ψ ⊖ 𝔄 Ψ (δ # Δ)
@ 𝔜• Ψ (δ # Δ)))
    using Cons
    by (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None, fastforce+)
  }
  then show ?case by blast
 qed
 thus ?thesis by blast
qed

lemma (in implication-logic) X-witness-msub:
 assumes mset (map snd Ψ) ⊆# mset Γ
    and mset (map snd Δ) ⊆# mset (map (uncurry (→)) Ψ @ Γ ⊖ (map snd
Ψ))
  shows mset (map snd (𝔛 Ψ Δ)) ⊆# mset Γ
proof −
 have mset (map snd (Δ ⊖ (𝔅 Ψ Δ))) ⊆# mset (Γ ⊖ (map snd Ψ))
  using assms second-component-diff-msub by blast
 moreover have mset (map snd (𝔄 Ψ Δ)) ⊆# mset (map snd Ψ)
  using first-component-msub
```

```
    by (simp add: image-mset-subseteq-mono)
  moreover have mset ((map snd Ψ) @ (Γ ⊖ map snd Ψ)) = mset Γ
    using assms(1)
    by simp
  moreover have image-mset snd (mset (𝔄 Ψ Δ)) + image-mset snd (mset (Δ ⊖
𝔅 Ψ Δ))
              = mset (map snd (𝔛 Ψ Δ))
      using X-witness-map-snd-decomposition by force
  ultimately
  show ?thesis
    by (metis (no-types) mset-append mset-map subset-mset.add-mono)
qed

lemma (in implication-logic) Y-component-msub:
  mset (map snd (𝔜• Ψ Δ)) ⊆# mset (map (uncurry (→)) (𝔛 Ψ Δ))
proof −
  have ∀ Ψ. mset (map snd (𝔜• Ψ Δ)) ⊆# mset (map (uncurry (→)) (𝔛 Ψ Δ))
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have mset (map snd (𝔜• Ψ (δ # Δ))) ⊆# mset (map (uncurry (→)) (𝔛 Ψ
(δ # Δ)))
        using Cons
        by (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None,
            simp, metis add-mset-add-single
                        mset-subset-eq-add-left
                        subset-mset.order-trans,
            fastforce)
    }
    then show ?case by blast
  qed
  thus ?thesis by blast
qed

lemma (in implication-logic) Y-witness-msub:
  assumes mset (map snd Ψ) ⊆# mset Γ
      and mset (map snd Δ) ⊆# mset (map (uncurry (→)) Ψ @ Γ ⊖ (map snd
Ψ))
    shows mset (map snd (𝔜 Ψ Δ)) ⊆#
          mset (map (uncurry (→)) (𝔛 Ψ Δ) @ Γ ⊖ map snd (𝔛 Ψ Δ))
proof −
  have A: image-mset snd (mset Ψ) ⊆# mset Γ using assms by simp
  have B: image-mset snd (mset (𝔄 Ψ Δ)) + image-mset snd (mset Δ − mset (𝔅
Ψ Δ)) ⊆# mset Γ
    using A X-witness-map-snd-decomposition assms(2) X-witness-msub by auto
```

```
  have mset Γ − image-mset snd (mset Ψ) = mset (Γ ⊖ map snd Ψ)
    by simp
  then have C: mset (map snd (Δ ⊖ 𝔅 Ψ Δ)) + image-mset snd (mset Ψ) ⊆# mset Γ
      using A by (metis (full-types) assms(2) second-component-diff-msub subset-mset.le-diff-conv2)
  have image-mset snd (mset (Ψ ⊖ 𝔄 Ψ Δ)) + image-mset snd (mset (𝔄 Ψ Δ)) = image-mset snd (mset Ψ)
    by (metis (no-types) image-mset-union
                         list-subtract-mset-homomorphism
                         first-component-msub
                         subset-mset.diff-add)
  then have image-mset snd (mset Ψ − mset (𝔄 Ψ Δ))
          + (image-mset snd (mset (𝔄 Ψ Δ)) + image-mset snd (mset Δ − mset (𝔅 Ψ Δ)))
        = mset (map snd (Δ ⊖ 𝔅 Ψ Δ)) + image-mset snd (mset Ψ)
    by (simp add: union-commute)
  then have image-mset snd (mset Ψ − mset (𝔄 Ψ Δ))
       ⊆# mset Γ − (image-mset snd (mset (𝔄 Ψ Δ)) + image-mset snd (mset Δ − mset (𝔅 Ψ Δ)))
      by (metis (no-types) B C subset-mset.le-diff-conv2)
  hence mset (map snd (Ψ ⊖ 𝔄 Ψ Δ)) ⊆# mset (Γ ⊖ map snd (𝔛 Ψ Δ))
    using assms X-witness-map-snd-decomposition
    by simp
  thus ?thesis
    using Y-component-msub
          Y-witness-map-snd-decomposition
    by (simp add: mset-subset-eq-mono-add union-commute)
qed

lemma (in classical-logic) X-witness-right-stronger-theory:
  map (uncurry (⊔)) Δ ≼ map (uncurry (⊔)) (𝔛 Ψ Δ)
proof −
  have ∀ Ψ. map (uncurry (⊔)) Δ ≼ map (uncurry (⊔)) (𝔛 Ψ Δ)
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have map (uncurry (⊔)) (δ # Δ) ≼ map (uncurry (⊔)) (𝔛 Ψ (δ # Δ))
      proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
        case True
        then show ?thesis
          using Cons
          by (simp add: stronger-theory-left-right-cons
                        trivial-implication)
      next
```

```
      case False
      from this obtain ψ where
        ψ: find (λψ. uncurry (→) ψ = snd δ) Ψ = Some ψ
           ψ ∈ set Ψ
           (fst ψ → snd ψ) = snd δ
        using find-Some-set-membership
              find-Some-predicate
        by fastforce
      let ?Ψ' = remove1 ψ Ψ
      let ?α = fst ψ
      let ?β = snd ψ
      let ?γ = fst δ
      have map (uncurry (⊔)) Δ ≼ map (uncurry (⊔)) (𝔛 ?Ψ' Δ)
        using Cons by simp
      moreover
      have (uncurry (⊔)) = (λ δ. fst δ ⊔ snd δ) by fastforce
      hence (uncurry (⊔)) δ = ?γ ⊔ (?α → ?β) using ψ(3) by fastforce
      moreover
      {
        fix α β γ
        have ⊢ (α → γ ⊔ β) → (γ ⊔ (α → β))
        proof −
          let ?φ = (⟨α⟩ → ⟨γ⟩ ⊔ ⟨β⟩) → (⟨γ⟩ ⊔ (⟨α⟩ → ⟨β⟩))
          have ∀ 𝔐. 𝔐 ⊨_prop ?φ by fastforce
          hence ⊢ ⦇ ?φ ⦈ using propositional-semantics by blast
          thus ?thesis by simp
        qed
      }
      hence ⊢ (?α → ?γ ⊔ ?β) → (?γ ⊔ (?α → ?β)) by simp
      ultimately
      show ?thesis using ψ
        by (simp add: stronger-theory-left-right-cons)
    qed
  }
  then show ?case by simp
 qed
 thus ?thesis by simp
qed

lemma (in classical-logic) Y-witness-left-stronger-theory:
  map (uncurry (⊔)) Ψ ≼ map (uncurry (⊔)) (𝔜 Ψ Δ)
proof −
  have ∀ Ψ. map (uncurry (⊔)) Ψ ≼ map (uncurry (⊔)) (𝔜 Ψ Δ)
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
```

```
    fix Ψ
    have map (uncurry (⊔)) Ψ ⪯ map (uncurry (⊔)) (𝔜 Ψ (δ # Δ))
    proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
      case True
      then show ?thesis using Cons by simp
    next
      case False
      from this obtain ψ where
        ψ: find (λψ. uncurry (→) ψ = snd δ) Ψ = Some ψ
           ψ ∈ set Ψ
           (uncurry (⊔)) ψ = fst ψ ⊔ snd ψ
        using find-Some-set-membership
        by fastforce
      let ?φ = fst ψ ⊔ (fst ψ → fst δ) → snd ψ
      let ?Ψ' = remove1 ψ Ψ
      have map (uncurry (⊔)) ?Ψ' ⪯ map (uncurry (⊔)) (𝔜 ?Ψ' Δ)
        using Cons by simp
      moreover
      {
        fix α β γ
        have ⊢ (α ⊔ (α → γ) → β) → (α ⊔ β)
        proof −
          let ?φ = (⟨α⟩ ⊔ (⟨α⟩ → ⟨γ⟩) → ⟨β⟩) → (⟨α⟩ ⊔ ⟨β⟩)
          have ∀ 𝔐. 𝔐 ⊨_prop ?φ by fastforce
          hence ⊢ ⦇ ?φ ⦈ using propositional-semantics by blast
          thus ?thesis by simp
        qed
      }
      hence ⊢ ?φ → (uncurry (⊔)) ψ using ψ(3) by auto
      ultimately
       have map (uncurry (⊔)) (ψ # ?Ψ') ⪯ (?φ # map (uncurry (⊔)) (𝔜 ?Ψ'
Δ))
        by (simp add: stronger-theory-left-right-cons)
      moreover
      from ψ have mset (map (uncurry (⊔)) (ψ # ?Ψ')) = mset (map (uncurry
(⊔)) Ψ)
        by (metis mset-map perm-remove)
      ultimately show ?thesis
        using stronger-theory-relation-alt-def ψ(1) by auto
    qed
   }
   then show ?case by blast
  qed
  thus ?thesis by blast
qed

lemma (in implication-logic) X-witness-second-component-diff-decomposition:
  mset (𝔛 Ψ Δ) = mset (𝔛• Ψ Δ @ Δ ⊖ 𝔅 Ψ Δ)
proof −
```

```
  have ∀ Ψ. mset (𝔛 Ψ Δ) = mset (𝔛• Ψ Δ @ Δ ⊖ 𝔅 Ψ Δ)
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have mset (𝔛 Ψ (δ # Δ)) =
           mset (𝔛• Ψ (δ # Δ) @ (δ # Δ) ⊖ 𝔅 Ψ (δ # Δ))
        using Cons
        by (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None,
        simp, metis add-mset-add-single second-component-msub subset-mset.diff-add-assoc2,
            fastforce)
    }
    then show ?case by blast
  qed
  thus ?thesis by blast
qed

lemma (in implication-logic) Y-witness-first-component-diff-decomposition:
  mset (𝔜 Ψ Δ) = mset (Ψ ⊖ 𝔄 Ψ Δ @ 𝔜• Ψ Δ)
proof −
  have ∀ Ψ. mset (𝔜 Ψ Δ) = mset (Ψ ⊖ 𝔄 Ψ Δ @ 𝔜• Ψ Δ)
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have mset (𝔜 Ψ (δ # Δ)) =
           mset (Ψ ⊖ 𝔄 Ψ (δ # Δ) @ 𝔜• Ψ (δ # Δ))
      using Cons
        by (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None, simp, fastforce)
    }
    then show ?case by blast
  qed
  thus ?thesis by blast
qed

lemma (in implication-logic) Y-witness-right-stronger-theory:
    map (uncurry (→)) Δ ≼ map (uncurry (→)) (𝔜 Ψ Δ ⊖ (Ψ ⊖ 𝔄 Ψ Δ) @ (Δ
⊖ 𝔅 Ψ Δ))
proof −
  let ?𝔣 = λΨ Δ. (Ψ ⊖ 𝔄 Ψ Δ)
  let ?𝔤 = λ Ψ Δ. (Δ ⊖ 𝔅 Ψ Δ)
  have ∀ Ψ. map (uncurry (→)) Δ ≼  map (uncurry (→)) (𝔜 Ψ Δ ⊖ ?𝔣 Ψ Δ @
?𝔤 Ψ Δ)
```

```
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    let ?δ = (uncurry (→)) δ
    {
      fix Ψ
      have map (uncurry (→)) (δ # Δ)
          ≼ map (uncurry (→)) (𝔜 Ψ (δ # Δ) ⊖ ?𝔣 Ψ (δ # Δ) @ ?𝔤 Ψ (δ # Δ))
      proof (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None)
        case True
        moreover
        from Cons have
          map (uncurry (→)) (δ # Δ) ≼ map (uncurry (→)) (δ # 𝔜 Ψ Δ ⊖ ?𝔣 Ψ
Δ @ ?𝔤 Ψ Δ)
          by (simp add: stronger-theory-left-right-cons trivial-implication)
        moreover
        have mset (map (uncurry (→)) (δ # 𝔜 Ψ Δ ⊖ ?𝔣 Ψ Δ @ ?𝔤 Ψ Δ))
          = mset (map (uncurry (→)) (𝔜 Ψ Δ ⊖ ?𝔣 Ψ Δ @ ((δ # Δ) ⊖ 𝔅 Ψ Δ)))
          by (simp,
              metis (no-types, lifting)
                    add-mset-add-single
                    image-mset-single
                    image-mset-union
                    second-component-msub
                    mset-subset-eq-multiset-union-diff-commute)
        moreover have
          ∀ Ψ Φ. Ψ ≼ Φ
              = (∃ Σ. map snd Σ = Ψ
                    ∧ mset (map fst Σ) ⊆# mset Φ
                    ∧ (∀ ξ. ξ ∉ set Σ ∨ ⊢ (uncurry (→) ξ)))
            by (simp add: Ball-def-raw stronger-theory-relation-def)
        moreover have
          ((uncurry (→) δ) # map (uncurry (→)) Δ)
           ≼ ((uncurry (→) δ) # map (uncurry (→)) (𝔜 Ψ Δ ⊖ (?𝔣 Ψ Δ))
             @ map (uncurry (→)) (?𝔤 Ψ Δ))
          using calculation by auto
        ultimately show ?thesis
          by (simp, metis union-mset-add-mset-right)
      next
        case False
        from this obtain ψ where
          ψ: find (λψ. uncurry (→) ψ = snd δ) Ψ = Some ψ
             uncurry (→) ψ = snd δ
          using find-Some-predicate
          by fastforce
        let ?α = fst ψ
        let ?β = fst δ
```

let *?γ* = *snd ψ*
have (λ *δ*. *fst δ* → *snd δ*) = *uncurry* (→) **by** *fastforce*
hence *?β* → *?α* → *?γ* = *uncurry* (→) *δ* **using** *ψ*(*2*) **by** *metis*
moreover
let *?A* = 𝔜 (*remove1 ψ* Ψ) Δ
let *?B* = 𝔄 (*remove1 ψ* Ψ) Δ
let *?C* = 𝔅 (*remove1 ψ* Ψ) Δ
let *?D* = *?A* ⊖ ((*remove1 ψ* Ψ) ⊖ *?B*)
have *mset* ((*remove1 ψ* Ψ) ⊖ *?B*) ⊆# *mset ?A*
using *Y-witness-first-component-diff-decomposition* **by** *simp*
{
assume *mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ)) ⊆# *mset* (𝔜 (*remove1 ψ* Ψ) Δ)
moreover have *B*: ∀ Φ Ψ. ∃ Δ. Ψ ⊆# Φ ⟶ Ψ + Δ = Φ
by (*metis subset-mset.le-iff-add*)
moreover obtain *f* **where**
A: *mset* (𝔜 (*remove1 ψ* Ψ) Δ)
− (*mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ)))
= *f* (*mset* (𝔜 (*remove1 ψ* Ψ) Δ))
(*mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ)))
by *blast*
ultimately obtain *g* **where**
B: ∀ *p*. *add-mset p* (*mset* (𝔜 (*remove1 ψ* Ψ) Δ))
− (*mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ)))
= *add-mset p*
(*g* (*mset* (𝔜 (*remove1 ψ* Ψ) Δ))
(*mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ))))
by (*metis add-diff-cancel-left′ union-mset-add-mset-right*)
have *g* (*mset* (𝔜 (*remove1 ψ* Ψ) Δ))
(*mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ)))
= *add-mset* (*fst ψ*, (*fst ψ* → *fst δ*) → *snd ψ*)
(*mset* (𝔜 (*remove1 ψ* Ψ) Δ))
− (*mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ)))
− {#(*fst ψ*, (*fst ψ* → *fst δ*) → *snd ψ*)#}
by (*simp add*: *B*)
then have *C*:
g (*mset* (𝔜 (*remove1 ψ* Ψ) Δ))
(*mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ)))
= *mset* (𝔜 (*remove1 ψ* Ψ) Δ)
− (*mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ)))
by *simp*
let *?S*$_1$ =
{# *x* → *y*.
(*x*, *y*) ∈# *add-mset* (*fst ψ*, (*fst ψ* → *fst δ*) → *snd ψ*)
(*mset* (𝔜 (*remove1 ψ* Ψ) Δ))
− (*mset* Ψ − *add-mset ψ* (*mset* (𝔄 (*remove1 ψ* Ψ) Δ)))
#}
let *?S*$_2$ =
add-mset

```
        (fst ψ → (fst ψ → fst δ) → snd ψ)
        {# x → y.
           (x, y) ∈# mset (𝔓 (remove1 ψ Ψ) Δ)
                    − (mset Ψ
                       − add-mset ψ (mset (𝔄 (remove1 ψ Ψ) Δ)))
         #}
      have ?S1 = ?S2
        using A C by (simp add: B)
    }
    hence mset (map (uncurry (→))
                (((?α, (?α → ?β) → ?γ) # ?A) ⊖ remove1 ψ (Ψ ⊖ ?B)
                 @ (remove1 δ ((δ # Δ) ⊖ ?C))))
          = mset ((?α → (?α → ?β) → ?γ) # map (uncurry (→)) (?D @ (Δ ⊖
?C)))
      using
        add-mset-add-single
        image-mset-add-mset
        prod.simps(2)
        subset-mset.diff-add-assoc2
        ‹mset (remove1 ψ Ψ ⊖ 𝔄 (remove1 ψ Ψ) Δ) ⊆# mset (𝔓 (remove1 ψ
Ψ) Δ)›
        by fastforce
    moreover
    have ⊢ (?α → (?α → ?β) → ?γ) → ?β → ?α → ?γ
    proof −
      let ?Γ = [(?α → (?α → ?β) → ?γ), ?β, ?α]
      have ?Γ :⊢ ?α → (?α → ?β) → ?γ
           ?Γ :⊢ ?α
        by (simp add: list-deduction-reflection)+
      hence ?Γ :⊢ (?α → ?β) → ?γ
        using list-deduction-modus-ponens by blast
      moreover have ?Γ :⊢ ?β
        by (simp add: list-deduction-reflection)
      hence ?Γ :⊢ ?α → ?β
       using axiom-k list-deduction-modus-ponens list-deduction-weaken by blast
      ultimately have ?Γ :⊢ ?γ
        using list-deduction-modus-ponens by blast
      thus ?thesis
        unfolding list-deduction-def by simp
    qed
    hence (?β → ?α → ?γ # map (uncurry (→)) Δ) ≼
          (?α → (?α → ?β) → ?γ # map (uncurry (→)) (?D @ (Δ ⊖ ?C)))
      using Cons stronger-theory-left-right-cons by blast
    ultimately show ?thesis
      using ψ by (simp add: stronger-theory-relation-alt-def)
  qed
 }
 then show ?case by blast
qed
```

```
  thus ?thesis by blast
qed

lemma (in implication-logic) xcomponent-ycomponent-connection:
  map (uncurry (→)) (𝔛• Ψ Δ) = map snd (𝔜• Ψ Δ)
proof −
  have ∀ Ψ. map (uncurry (→)) (𝔛• Ψ Δ) = map snd (𝔜• Ψ Δ)
  proof (induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have map (uncurry (→)) (𝔛• Ψ (δ # Δ)) = map snd (𝔜• Ψ (δ # Δ))
        using Cons
        by (cases find (λ ψ. (uncurry (→)) ψ = snd δ) Ψ = None, simp, fastforce)
    }
    then show ?case by blast
  qed
  thus ?thesis by blast
qed

lemma (in classical-logic) xwitness-ywitness-measure-deduction-intro:
  assumes mset (map snd Ψ) ⊆# mset Γ
      and mset (map snd Δ) ⊆# mset (map (uncurry (→)) Ψ @ Γ ⊖ (map snd
Ψ))
      and map (uncurry (→)) Δ @ (map (uncurry (→)) Ψ @ Γ ⊖ map snd Ψ) ⊖
map snd Δ $⊢ Φ
          (is ?Γ₀ $⊢ Φ)
        shows map (uncurry (→)) (𝔜 Ψ Δ) @
              (map (uncurry (→)) (𝔛 Ψ Δ) @ Γ ⊖ map snd (𝔛 Ψ Δ)) ⊖
               map snd (𝔜 Ψ Δ) $⊢ Φ
          (is ?Γ $⊢ Φ)
proof −
  let ?A = map (uncurry (→)) (𝔜 Ψ Δ)
  let ?B = map (uncurry (→)) (𝔛 Ψ Δ)
  let ?C = Ψ ⊖ 𝔄 Ψ Δ
  let ?D = map (uncurry (→)) ?C
  let ?E = Δ ⊖ 𝔅 Ψ Δ
  let ?F = map (uncurry (→)) ?E
  let ?G = map snd (𝔅 Ψ Δ)
  let ?H = map (uncurry (→)) (𝔛• Ψ Δ)
  let ?I = 𝔄 Ψ Δ
  let ?J = map snd (𝔛 Ψ Δ)
  let ?K = map snd (𝔜 Ψ Δ)
  have mset (map (uncurry (→)) (𝔜 Ψ Δ ⊖ ?C @ ?E)) = mset (?A ⊖ ?D @ ?F)
    by (simp add: Y-witness-first-component-diff-decomposition)
  hence (map (uncurry (→)) Δ) ≼ (?A ⊖ ?D @ ?F)
```

using *Y-witness-right-stronger-theory*
stronger-theory-relation-alt-def
by (*simp*, *metis* (*no-types*, *lifting*))
hence *?*$\Gamma_0 \preceq ((?A \ominus ?D \mathbin{@} ?F) \mathbin{@} (map\ (uncurry\ (\rightarrow))\ \Psi \mathbin{@} \Gamma \ominus map\ snd\ \Psi) \ominus map\ snd\ \Delta)$
using *stronger-theory-combine stronger-theory-reflexive* **by** *blast*
moreover
have ♠: $mset\ ?G \subseteq\# mset\ (map\ (uncurry\ (\rightarrow))\ \Psi)$
$mset\ (\mathfrak{B}\ \Psi\ \Delta) \subseteq\# mset\ \Delta$
$mset\ (map\ snd\ ?E) \subseteq\# mset\ (\Gamma \ominus map\ snd\ \Psi)$
$mset\ (map\ (uncurry\ (\rightarrow))\ \Psi \ominus ?G) = mset\ ?D$
$mset\ ?D \subseteq\# mset\ ?A$
$mset\ (map\ snd\ ?I) \subseteq\# mset\ (map\ snd\ \Psi)$
$mset\ (map\ snd\ ?I) \subseteq\# mset\ \Gamma$
$mset\ (map\ snd\ (?I \mathbin{@} ?E)) = mset\ ?J$
using *second-component-msub*
second-component-diff-msub
second-component-snd-projection-msub
first-component-second-component-mset-connection
X-witness-map-snd-decomposition

by (*simp*,
simp,
metis assms(*2*),
simp add: *image-mset-Diff first-component-msub*,
simp add: *Y-witness-first-component-diff-decomposition*,
simp add: *image-mset-subseteq-mono first-component-msub*,
metis assms(*1*) *first-component-msub map-monotonic subset-mset.dual-order.trans*,
simp)
hence $mset\ \Delta - mset\ (\mathfrak{B}\ \Psi\ \Delta) + mset\ (\mathfrak{B}\ \Psi\ \Delta) = mset\ \Delta$
by *simp*
hence ♡: $\{\#x \rightarrow y.\ (x, y) \in\# mset\ \Psi\#\} + (mset\ \Gamma - image\text{-}mset\ snd\ (mset\ \Psi))$
$- image\text{-}mset\ snd\ (mset\ \Delta)$
$= \{\#x \rightarrow y.\ (x, y) \in\# mset\ \Psi\#\} + (mset\ \Gamma - image\text{-}mset\ snd\ (mset\ \Psi))$
$- image\text{-}mset\ snd\ (mset\ \Delta - mset\ (\mathfrak{B}\ \Psi\ \Delta))$
$- image\text{-}mset\ snd\ (mset\ (\mathfrak{B}\ \Psi\ \Delta))$
$image\text{-}mset\ snd\ (mset\ \Psi - mset\ (\mathfrak{A}\ \Psi\ \Delta)) + image\text{-}mset\ snd\ (mset\ (\mathfrak{A}\ \Psi\ \Delta))$
$= image\text{-}mset\ snd\ (mset\ \Psi)$
using ♠
by (*metis* (*no-types*) *diff-diff-add-mset image-mset-union*,
metis (*no-types*) *image-mset-union first-component-msub subset-mset.diff-add*)
then have $mset\ \Gamma - image\text{-}mset\ snd\ (mset\ \Psi)$
$- image\text{-}mset\ snd\ (mset\ \Delta - mset\ (\mathfrak{B}\ \Psi\ \Delta))$
$= mset\ \Gamma - (image\text{-}mset\ snd\ (mset\ \Psi - mset\ (\mathfrak{A}\ \Psi\ \Delta))$
$+ image\text{-}mset\ snd\ (mset\ (\mathfrak{X}\ \Psi\ \Delta)))$
using ♠ **by** (*simp*, *metis* (*full-types*) *diff-diff-add-mset*)
hence $mset\ ((map\ (uncurry\ (\rightarrow))\ \Psi \mathbin{@} \Gamma \ominus map\ snd\ \Psi) \ominus map\ snd\ \Delta)$

```
    = mset (?D @ (Γ ⊖ ?J) ⊖ map snd ?C)
  using ♡ ♠ by (simp, metis (no-types) add.commute subset-mset.add-diff-assoc)
 ultimately have ?Γ₀ ≼ ((?A ⊖ ?D @ ?F) @ ?D @ (Γ ⊖ ?J) ⊖ map snd ?C)
  unfolding stronger-theory-relation-alt-def
  by simp
 moreover
 have mset ?F = mset (?B ⊖ ?H)
     mset ?D ⊆# mset ?A
     mset (map snd (Ψ ⊖ ?I)) ⊆# mset (Γ ⊖ ?J)
  by (simp add: X-witness-second-component-diff-decomposition,
      simp add: Y-witness-first-component-diff-decomposition,
      simp, metis (no-types, lifting)
            ♡(2) ♠(8) add.assoc assms(1) assms(2) image-mset-union
            X-witness-msub merge-witness-msub-intro
            second-component-merge-witness-snd-projection
            mset-map
            subset-mset.le-diff-conv2
            union-code)
 hence mset ((?A ⊖ ?D @ ?F) @ ?D @ (Γ ⊖ ?J) ⊖ map snd ?C)
     = mset (?A @ (?B ⊖ ?H @ Γ ⊖ ?J) ⊖ map snd ?C)
      mset ?H ⊆# mset ?B
      {#x → y. (x, y) ∈# mset (𝔛• Ψ Δ)#} = mset (map snd (𝔜• Ψ Δ))
  by (simp add: subset-mset.diff-add-assoc,
      simp add: X-witness-second-component-diff-decomposition,
      metis xcomponent-ycomponent-connection mset-map uncurry-def)
 hence mset ((?A ⊖ ?D @ ?F) @ ?D @ (Γ ⊖ ?J) ⊖ map snd ?C)
     = mset (?A @ (?B @ Γ ⊖ ?J) ⊖ (?H @ map snd ?C))
      {#x → y. (x, y) ∈# mset (𝔛• Ψ Δ)#} + image-mset snd (mset Ψ − mset
(𝔄 Ψ Δ))
     = mset (map snd (𝔜 Ψ Δ))
   using Y-witness-map-snd-decomposition
   by (simp add: subset-mset.diff-add-assoc, force)
 hence mset ((?A ⊖ ?D @ ?F) @ ?D @ (Γ ⊖ ?J) ⊖ map snd ?C)
     = mset (?A @ (?B @ Γ ⊖ ?J) ⊖ ?K)
   by (simp)
 ultimately have ?Γ₀ ≼ (?A @ (?B @ Γ ⊖ ?J) ⊖ ?K)
   unfolding stronger-theory-relation-alt-def
   by metis
 thus ?thesis
   using assms(3) measure-stronger-theory-left-monotonic
   by blast
qed

lemma (in classical-logic) measure-cons-cons-right-permute:
 assumes Γ $⊢ (φ # ψ # Φ)
 shows Γ $⊢ (ψ # φ # Φ)
proof −
 from assms obtain Ψ where Ψ:
   mset (map snd Ψ) ⊆# mset Γ
```

```
    map (uncurry (⊔)) Ψ :⊢ φ
    map (uncurry (→)) Ψ @ Γ ⊖ (map snd Ψ) $⊢ (ψ # Φ)
    by fastforce
  let ?Γ₀ = map (uncurry (→)) Ψ @ Γ ⊖ (map snd Ψ)
  from Ψ(3) obtain Δ where Δ:
    mset (map snd Δ) ⊆# mset ?Γ₀
    map (uncurry (⊔)) Δ :⊢ ψ
    (map (uncurry (→)) Δ @ ?Γ₀ ⊖ (map snd Δ)) $⊢ Φ
    using measure-deduction.simps(2) by blast
  let ?Ψ' = 𝔛 Ψ Δ
  let ?Γ₁ = map (uncurry (→)) ?Ψ' @ Γ ⊖ (map snd ?Ψ')
  let ?Δ' = 𝔜 Ψ Δ
  have (map (uncurry (→)) ?Δ' @ ?Γ₁ ⊖ (map snd ?Δ')) $⊢ Φ
      map (uncurry (⊔)) Ψ ≼ map (uncurry (⊔)) ?Δ'
    using Ψ(1) Δ(1) Δ(3)
          xwitness-ywitness-measure-deduction-intro
          Y-witness-left-stronger-theory
    by auto
  hence ?Γ₁ $⊢ (φ # Φ)
    using Ψ(1) Ψ(2) Δ(1)
          Y-witness-msub measure-deduction.simps(2)
          stronger-theory-deduction-monotonic
    by blast
  thus ?thesis
    using Ψ(1) Δ(1) Δ(2)
          X-witness-msub
          X-witness-right-stronger-theory
          measure-deduction.simps(2)
          stronger-theory-deduction-monotonic
    by blast
qed

lemma (in classical-logic) measure-cons-remove1:
  assumes φ ∈ set Φ
    shows Γ $⊢ Φ = Γ $⊢ (φ # (remove1 φ Φ))
proof −
  from ‹φ ∈ set Φ›
  have ∀ Γ. Γ $⊢ Φ = Γ $⊢ (φ # (remove1 φ Φ))
  proof (induct Φ)
    case Nil
    then show ?case by simp
  next
    case (Cons χ Φ)
    {
      fix Γ
      have Γ $⊢ (χ # Φ) = Γ $⊢ (φ # (remove1 φ (χ # Φ)))
      proof (cases χ = φ)
        case True
        then show ?thesis by simp
```

```
    next
      case False
      hence φ ∈ set Φ
        using Cons.prems by simp
      with Cons.hyps have Γ $⊢ (χ # Φ) = Γ $⊢ (χ # φ # (remove1 φ Φ))
        by fastforce
      hence Γ $⊢ (χ # Φ) = Γ $⊢ (φ # χ # (remove1 φ Φ))
        using measure-cons-cons-right-permute by blast
      then show ?thesis using ‹χ ≠ φ› by simp
    qed
  }
  then show ?case by blast
 qed
 thus ?thesis using assms by blast
qed

lemma (in classical-logic) witness-stronger-theory:
 assumes mset (map snd Ψ) ⊆# mset Γ
 shows (map (uncurry (→)) Ψ @ Γ ⊖ (map snd Ψ)) ≼ Γ
proof −
 have ∀ Γ. mset (map snd Ψ) ⊆# mset Γ ⟶ (map (uncurry (→)) Ψ @ Γ ⊖
(map snd Ψ)) ≼ Γ
 proof (induct Ψ)
   case Nil
   then show ?case by simp
 next
   case (Cons ψ Ψ)
   let ?γ = snd ψ
   {
     fix Γ
     assume mset (map snd (ψ # Ψ)) ⊆# mset Γ
     hence mset (map snd Ψ) ⊆# mset (remove1 (snd ψ) Γ)
       by (simp add: insert-subset-eq-iff)
     with Cons have
      (map (uncurry (→)) Ψ @ (remove1 (snd ψ) Γ) ⊖ (map snd Ψ)) ≼ (remove1
?γ Γ)
       by blast
     hence (map (uncurry (→)) Ψ @ Γ ⊖ (map snd (ψ # Ψ))) ≼ (remove1 ?γ Γ)
       by (simp add: stronger-theory-relation-alt-def)
     moreover
     have (uncurry (→)) = (λ ψ. fst ψ → snd ψ)
       by fastforce
     hence ⊢ ?γ → uncurry (→) ψ
       using axiom-k by simp
     ultimately have
      (map (uncurry (→)) (ψ # Ψ) @ Γ ⊖ (map snd (ψ # Ψ))) ≼ (?γ # (remove1
?γ Γ))
       using stronger-theory-left-right-cons by auto
     hence (map (uncurry (→)) (ψ # Ψ) @ Γ ⊖ (map snd (ψ # Ψ))) ≼ Γ
```

```
    using stronger-theory-relation-alt-def
        ‹mset (map snd (ψ # Ψ)) ⊆# mset Γ›
        mset-subset-eqD
    by fastforce
  }
  then show ?case by blast
 qed
 thus ?thesis using assms by blast
qed

lemma (in classical-logic) measure-msub-weaken:
 assumes mset Ψ ⊆# mset Φ
   and Γ $⊢ Φ
  shows Γ $⊢ Ψ
proof −
 have ∀ Ψ Γ. mset Ψ ⊆# mset Φ ⟶ Γ $⊢ Φ ⟶ Γ $⊢ Ψ
 proof (induct Φ)
  case Nil
  then show ?case by simp
 next
  case (Cons φ Φ)
  {
   fix Ψ Γ
   assume mset Ψ ⊆# mset (φ # Φ)
        Γ $⊢ (φ # Φ)
   hence Γ $⊢ Φ
    using measure-deduction.simps(2)
        measure-stronger-theory-left-monotonic
        witness-stronger-theory
    by blast
   have Γ $⊢ Ψ
   proof (cases φ ∈ set Ψ)
    case True
    hence mset (remove1 φ Ψ) ⊆# mset Φ
     using ‹mset Ψ ⊆# mset (φ # Φ)›
         subset-eq-diff-conv
     by force
    hence ∀ Γ. Γ $⊢ Φ ⟶ Γ $⊢ (remove1 φ Ψ)
     using Cons by blast
    hence Γ $⊢ (φ # (remove1 φ Ψ))
     using ‹Γ $⊢ (φ # Φ)› by fastforce
    then show ?thesis
     using ‹φ ∈ set Ψ›
         measure-cons-remove1
     by blast
   next
    case False
    have mset Ψ ⊆# mset Φ + add-mset φ (mset [])
     using ‹mset Ψ ⊆# mset (φ # Φ)› by auto
```

```
      hence mset Ψ ⊆# mset Φ
        by (metis (no-types) False
                          diff-single-trivial
                          in-multiset-in-set mset.simps(1)
                          subset-eq-diff-conv)
      then show ?thesis
        using ‹Γ $⊢ Φ› Cons
        by blast
    qed
  }
  then show ?case by blast
 qed
 with assms show ?thesis by blast
qed

lemma (in classical-logic) measure-stronger-theory-right-antitonic:
 assumes Ψ ⪯ Φ
    and Γ $⊢ Φ
  shows Γ $⊢ Ψ
proof −
 have ∀ Ψ Γ. Ψ ⪯ Φ ⟶ Γ $⊢ Φ ⟶ Γ $⊢ Ψ
 proof (induct Φ)
  case Nil
  then show ?case
    using measure-deduction.simps(1)
          stronger-theory-empty-list-intro
    by blast
 next
  case (Cons φ Φ)
  {
   fix Ψ Γ
   assume Γ $⊢ (φ # Φ)
          Ψ ⪯ (φ # Φ)
   from this obtain Σ where
     Σ: map snd Σ = Ψ
        mset (map fst Σ) ⊆# mset (φ # Φ)
        ∀ (φ,ψ)∈set Σ. ⊢ φ → ψ
     unfolding stronger-theory-relation-def
     by auto
   hence Γ $⊢ Ψ
   proof (cases φ ∈ set (map fst Σ))
     case True
     from this obtain ψ where (φ,ψ) ∈ set Σ
       by (induct Σ, simp, fastforce)
     hence A: mset (map snd (remove1 (φ, ψ) Σ)) = mset (remove1 ψ Ψ)
       and B: mset (map fst (remove1 (φ, ψ) Σ)) ⊆# mset Φ
       using Σ remove1-pairs-list-projections-snd
             remove1-pairs-list-projections-fst
             subset-eq-diff-conv
```

```
      by fastforce+
    have ∀(φ,ψ)∈set (remove1 (φ, ψ) Σ). ⊢ φ → ψ
      using Σ(3) by fastforce+
    hence (remove1 ψ Ψ) ≼ Φ
      unfolding stronger-theory-relation-alt-def using A B by blast
    moreover
    from ‹Γ $⊢ (φ # Φ)› obtain Δ where
      Δ: mset (map snd Δ) ⊆# mset Γ
         map (uncurry (⊔)) Δ :⊢ φ
         (map (uncurry (→)) Δ @ Γ ⊖ (map snd Δ)) $⊢ Φ
      by auto
    ultimately have (map (uncurry (→)) Δ @ Γ ⊖ (map snd Δ)) $⊢ remove1
ψ Ψ
      using Cons by blast
    moreover have map (uncurry (⊔)) Δ :⊢ ψ
      using Δ(2) Σ(3) ‹(φ,ψ) ∈ set Σ›
            list-deduction-weaken
            list-deduction-modus-ponens
      by blast
    ultimately have ‹Γ $⊢ (ψ # (remove1 ψ Ψ))›
      using Δ(1) by auto
    moreover from ‹(φ,ψ) ∈ set Σ› Σ(1) have ψ ∈ set Ψ
      by force
    hence mset Ψ ⊆# mset (ψ # (remove1 ψ Ψ))
      by auto
    ultimately show ?thesis using measure-msub-weaken by blast
  next
    case False
    hence mset (map fst Σ) ⊆# mset Φ
      using Σ(2)
      by (simp,
        metis add-mset-add-single
              diff-single-trivial
              mset-map set-mset-mset
              subset-eq-diff-conv)
    hence Ψ ≼ Φ
      using Σ(1) Σ(3)
      unfolding stronger-theory-relation-def
      by auto
    moreover from ‹Γ $⊢ (φ # Φ)› have Γ $⊢ Φ
      using measure-deduction.simps(2)
        measure-stronger-theory-left-monotonic
        witness-stronger-theory
      by blast
    ultimately show ?thesis using Cons by blast
  qed
 }
 then show ?case by blast
qed
```

thus *?thesis* **using** *assms* **by** *blast*
qed

lemma (**in** *classical-logic*) *measure-witness-right-split*:
assumes *mset* (*map snd* Ψ) ⊆# *mset* Φ
shows Γ $⊢ (*map* (*uncurry* (⊔)) Ψ @ *map* (*uncurry* (→)) Ψ @ Φ ⊖ (*map snd* Ψ)) = Γ $⊢ Φ
proof −
have ∀ Γ Φ. *mset* (*map snd* Ψ) ⊆# *mset* Φ ⟶
Γ $⊢ Φ = Γ $⊢ (*map* (*uncurry* (⊔)) Ψ @ *map* (*uncurry* (→)) Ψ @ Φ ⊖ (*map snd* Ψ))
proof (*induct* Ψ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* ψ Ψ)
{
fix Γ Φ
let *?χ* = *fst* ψ
let *?φ* = *snd* ψ
let *?Φ′* = *map* (*uncurry* (⊔)) (ψ # Ψ) @
map (*uncurry* (→)) (ψ # Ψ) @
Φ ⊖ *map snd* (ψ # Ψ)
let *?Φ*$_0$ = *map* (*uncurry* (⊔)) Ψ @
map (*uncurry* (→)) Ψ @
(*remove1 ?φ* Φ) ⊖ *map snd* Ψ
assume *mset* (*map snd* (ψ # Ψ)) ⊆# *mset* Φ
hence *mset* (*map snd* Ψ) ⊆# *mset* (*remove1 ?φ* Φ)
mset (*?φ* # *remove1 ?φ* Φ) = *mset* Φ
by (*simp add*: *insert-subset-eq-iff*)+
hence Γ $⊢ Φ = Γ $⊢ (*?φ* # *remove1 ?φ* Φ)
∀ Γ. Γ $⊢ (*remove1 ?φ* Φ) = Γ $⊢ *?Φ*$_0$
by (*metis list.set-intros*(*1*) *measure-cons-remove1 set-mset-mset*,
metis Cons.hyps)
moreover
have (*uncurry* (⊔)) = (λ ψ. *fst* ψ ⊔ *snd* ψ)
(*uncurry* (→)) = (λ ψ. *fst* ψ → *snd* ψ)
by *fastforce*+
hence *mset ?Φ′* ⊆# *mset* (*?χ* ⊔ *?φ* # *?χ* → *?φ* # *?Φ*$_0$)
mset (*?χ* ⊔ *?φ* # *?χ* → *?φ* # *?Φ*$_0$) ⊆# *mset ?Φ′*
(**is** *mset ?X* ⊆# *mset ?Y*)
by *fastforce*+
hence Γ $⊢ *?Φ′* = Γ $⊢ (*?φ* # *?Φ*$_0$)
using *measure-formula-right-split*
measure-msub-weaken
by *blast*
ultimately have Γ $⊢ Φ = Γ $⊢ *?Φ′*
by *fastforce*
}

```
    then show ?case by blast
  qed
  with assms show ?thesis by blast
qed

primrec (in classical-logic)
  submerge-witness :: ('a × 'a) list ⇒ ('a × 'a) list ⇒ ('a × 'a) list (𝔈)
  where
    𝔈 Σ [] = map (λ σ. (⊥, (uncurry (⊔)) σ)) Σ
  | 𝔈 Σ (δ # Δ) =
      (case find (λ σ. (uncurry (→)) σ = snd δ) Σ of
          None ⇒ 𝔈 Σ Δ
        | Some σ ⇒ (fst σ, (fst δ ⊓ fst σ) ⊔ snd σ) # (𝔈 (remove1 σ Σ) Δ))

lemma (in classical-logic) submerge-witness-stronger-theory-left:
   map (uncurry (⊔)) Σ ≼ map (uncurry (⊔)) (𝔈 Σ Δ)
proof −
  have ∀ Σ. map (uncurry (⊔)) Σ ≼ map (uncurry (⊔)) (𝔈 Σ Δ)
  proof (induct Δ)
    case Nil
    {
      fix Σ
      {
        fix φ
        have ⊢ (⊥ ⊔ φ) → φ
          unfolding disjunction-def
         using ex-falso-quodlibet modus-ponens excluded-middle-elimination by blast
      }
      note tautology = this
      have map (uncurry (⊔)) Σ ≼ map (uncurry (⊔)) (𝔈 Σ [])
        by (induct Σ,
            simp,
            simp add: stronger-theory-left-right-cons tautology)
    }
    then show ?case by auto
  next
    case (Cons δ Δ)
    {
      fix Σ
      have map (uncurry (⊔)) Σ ≼ map (uncurry (⊔)) (𝔈 Σ (δ # Δ))
      proof (cases find (λ σ. (uncurry (→)) σ = snd δ) Σ = None)
        case True
        then show ?thesis using Cons by simp
      next
        case False
        from this obtain σ where
          σ: find (λσ. uncurry (→) σ = snd δ) Σ = Some σ
             uncurry (→) σ = snd δ
             σ ∈ set Σ
```

```
      using find-Some-predicate find-Some-set-membership
      by fastforce
    {
      fix α β γ
      have ⊢ (α ⊔ (γ ⊓ α) ⊔ β) → (α ⊔ β)
      proof −
        let ?φ = (⟨α⟩ ⊔ (⟨γ⟩ ⊓ ⟨α⟩) ⊔ ⟨β⟩) → (⟨α⟩ ⊔ ⟨β⟩)
        have ∀ 𝔐. 𝔐 ⊨_prop ?φ by fastforce
        hence ⊢ ⦇ ?φ ⦈ using propositional-semantics by blast
        thus ?thesis by simp
      qed
    }
    note tautology = this
    let ?α = fst σ
    let ?β = snd σ
    let ?γ = fst δ
    have (uncurry (⊔)) = (λ σ. fst σ ⊔ snd σ) by fastforce
    hence (uncurry (⊔)) σ = ?α ⊔ ?β by simp
   hence A: ⊢ (?α ⊔ (?γ ⊓ ?α) ⊔ ?β) → (uncurry (⊔)) σ using tautology by
simp
    moreover
    have map (uncurry (⊔)) (remove1 σ Σ)
        ≼ map (uncurry (⊔)) (𝔈 (remove1 σ Σ) Δ)
      using Cons by simp
    ultimately have A:
      map (uncurry (⊔)) (σ # (remove1 σ Σ))
       ≼ (?α ⊔ (?γ ⊓ ?α) ⊔ ?β # map (uncurry (⊔)) (𝔈 (remove1 σ Σ) Δ))
        using stronger-theory-left-right-cons by fastforce
    from σ(3) have mset Σ = mset (σ # (remove1 σ Σ))
      by simp
     hence mset (map (uncurry (⊔)) Σ) = mset (map (uncurry (⊔)) (σ #
(remove1 σ Σ)))
      by (metis mset-map)
   hence B: map (uncurry (⊔)) Σ ≼ map (uncurry (⊔)) (σ # (remove1 σ Σ))
      by (simp add: msub-stronger-theory-intro)
    have (  fst σ
          ⊔ (fst δ ⊓ fst σ)
          ⊔ snd σ # map (λ(x, y). x ⊔ y) (𝔈 (remove1 σ Σ) Δ)) ≽ map (λ(x,
y). x ⊔ y) Σ
      by (metis
           (no-types, lifting)
           A B
           stronger-theory-transitive
           uncurry-def)
    thus ?thesis using A B σ by simp
  qed
 }
 then show ?case by auto
qed
```

```
  thus ?thesis by blast
qed

lemma (in classical-logic) submerge-witness-msub:
  mset (map snd (𝔈 Σ Δ)) ⊆# mset (map (uncurry (⊔)) (𝔍 Σ Δ))
proof −
  have ∀ Σ. mset (map snd (𝔈 Σ Δ)) ⊆# mset (map (uncurry (⊔)) (𝔍 Σ Δ))
  proof (induct Δ)
    case Nil
    {
      fix Σ
      have mset (map snd (𝔈 Σ [])) ⊆#
           mset (map (uncurry (⊔)) (𝔍 Σ []))
        by (induct Σ, simp+)
    }
    then show ?case by blast
  next
    case (Cons δ Δ)
    {
      fix Σ
      have mset (map snd (𝔈 Σ (δ # Δ))) ⊆#
           mset (map (uncurry (⊔)) (𝔍 Σ (δ # Δ)))
        using Cons
        by (cases find (λ σ. (uncurry (→)) σ = snd δ) Σ = None,
            simp,
            meson diff-subset-eq-self
                  insert-subset-eq-iff
                  mset-subset-eq-add-mset-cancel
                  subset-mset.dual-order.trans,
            fastforce)
    }
    then show ?case by blast
  qed
  thus ?thesis by blast
qed

lemma (in classical-logic) submerge-witness-stronger-theory-right:
   map (uncurry (⊔)) Δ
 ≼ (map (uncurry (→)) (𝔈 Σ Δ) @ map (uncurry (⊔)) (𝔍 Σ Δ) ⊖ map snd (𝔈 Σ
Δ))
proof −
  have ∀ Σ. map (uncurry (⊔)) Δ
         ≼ (map (uncurry (→)) (𝔈 Σ Δ) @ map (uncurry (⊔)) (𝔍 Σ Δ)  ⊖ map
snd (𝔈 Σ Δ))
  proof(induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
```

```
{
  fix Σ
  have map (uncurry (⊔)) (δ # Δ) ≼
      (  map (uncurry (→)) (𝔈 Σ (δ # Δ))
       @ map (uncurry (⊔)) (𝔍 Σ (δ # Δ))
         ⊖ map snd (𝔈 Σ (δ # Δ)))
  proof (cases find (λ σ. (uncurry (→)) σ = snd δ) Σ = None)
    case True
    from Cons obtain Φ where Φ:
      map snd Φ = map (uncurry (⊔)) Δ
      mset (map fst Φ) ⊆#
        mset (map (uncurry (→)) (𝔈 Σ Δ)
             @ map (uncurry (⊔)) (𝔍 Σ Δ) ⊖ map snd (𝔈 Σ Δ))
      ∀(γ, σ)∈set Φ. ⊢ γ → σ
      unfolding stronger-theory-relation-def
      by fastforce
    let ?Φ' = (uncurry (⊔) δ, (uncurry (⊔)) δ) # Φ
    have map snd ?Φ' = map (uncurry (⊔)) (δ # Δ) using Φ(1) by simp
    moreover
    from Φ(2) have A:
      image-mset fst (mset Φ)
    ⊆# {#x → y. (x, y) ∈# mset (𝔈 Σ Δ)#}
       + ({#x ⊔ y. (x, y) ∈# mset (𝔍 Σ Δ)#} − image-mset snd (mset (𝔈 Σ
Δ)))
      by simp
     have image-mset snd (mset (𝔈 Σ Δ)) ⊆# {#x ⊔ y. (x, y) ∈# mset (𝔍 Σ
Δ)#}
      using submerge-witness-msub by force
    then have B: {#case δ of (x, xa) ⇒ x ⊔ xa#}
            ⊆# add-mset (case δ of (x, xa) ⇒ x ⊔ xa)
                  {#x ⊔ y. (x, y) ∈# mset (𝔍 Σ Δ)#} − image-mset snd
(mset (𝔈 Σ Δ))
      by (metis add-mset-add-single subset-mset.le-add-diff)
    have add-mset (case δ of (x, xa) ⇒ x ⊔ xa) {#x ⊔ y. (x, y) ∈# mset (𝔍 Σ
Δ)#}
         − image-mset snd (mset (𝔈 Σ Δ)) − {#case δ of (x, xa) ⇒ x ⊔ xa#}
        = {#x ⊔ y. (x, y) ∈# mset (𝔍 Σ Δ)#} − image-mset snd (mset (𝔈 Σ
Δ))
      by force
     then have add-mset (case δ of (x, xa) ⇒ x ⊔ xa) (image-mset fst (mset
Φ))
           − (add-mset (case δ of (x, xa) ⇒ x ⊔ xa) {#x ⊔ y. (x, y) ∈# mset
(𝔍 Σ Δ)#}
            − image-mset snd (mset (𝔈 Σ Δ)))
         ⊆# {#x → y. (x, y) ∈# mset (𝔈 Σ Δ)#}
      using A B by (metis (no-types) add-mset-add-single
                          subset-eq-diff-conv
                          subset-mset.diff-diff-right)
    hence add-mset (case δ of (x, xa) ⇒ x ⊔ xa) (image-mset fst (mset Φ))
```

```
        ⊆# {#x → y. (x, y) ∈# mset (𝔈 Σ Δ)#}
          + (add-mset (case δ of (x, xa) ⇒ x ⊔ xa) {#x ⊔ y. (x, y) ∈# mset (𝔍
Σ Δ)#}
            − image-mset snd (mset (𝔈 Σ Δ)))
        using subset-eq-diff-conv by blast
      hence
        mset (map fst ?Φ′) ⊆#
          mset (map (uncurry (→)) (𝔈 Σ (δ # Δ))
              @ map (uncurry (⊔)) (𝔍 Σ (δ # Δ))
                ⊖ map snd (𝔈 Σ (δ # Δ)))
        using True Φ(2)
        by simp
      moreover have ∀(γ, σ)∈set ?Φ′. ⊢ γ → σ
        using Φ(3) trivial-implication by auto
      ultimately show ?thesis
        unfolding stronger-theory-relation-def
        by blast
    next
      case False
      from this obtain σ where
        σ: find (λσ. uncurry (→) σ = snd δ) Σ = Some σ
          uncurry (→) σ = snd δ
        using find-Some-predicate
        by fastforce
      moreover from Cons have
        map (uncurry (⊔)) Δ ≼
        (map (uncurry (→)) (𝔈 (remove1 σ Σ) Δ) @
          remove1 ((fst δ ⊓ fst σ) ⊔ snd σ)
           (((fst δ ⊓ fst σ) ⊔ snd σ # map (uncurry (⊔)) (𝔍 (remove1 σ Σ) Δ))
             ⊖ map snd (𝔈 (remove1 σ Σ) Δ)))
        unfolding stronger-theory-relation-alt-def
        by simp
      moreover
      {
        fix α β γ
        have ⊢ (α → ((γ ⊓ α) ⊔ β)) → (γ ⊔ (α → β))
        proof −
          let ?φ = (⟨α⟩ → ((⟨γ⟩ ⊓ ⟨α⟩) ⊔ ⟨β⟩)) → (⟨γ⟩ ⊔ (⟨α⟩ → ⟨β⟩))
          have ∀𝔐. 𝔐 ⊨_prop ?φ by fastforce
          hence ⊢ ⦇ ?φ ⦈ using propositional-semantics by blast
          thus ?thesis by simp
        qed
      }
      note tautology = this
      let ?α = fst σ
      let ?β = snd σ
      let ?γ = fst δ
      have (λ δ. uncurry (⊔) δ) = (λ δ. fst δ ⊔ snd δ)
          (λ σ. uncurry (→) σ) = (λ σ. fst σ → snd σ) by fastforce+
```

```
      hence (uncurry (⊔) δ) = (?γ ⊔ (?α → ?β)) using σ(2) by simp
      hence ⊢ (?α → ((?γ ⊓ ?α) ⊔ ?β)) → (uncurry (⊔) δ) using tautology by
auto
      ultimately show ?thesis
        using stronger-theory-left-right-cons
        by fastforce
    qed
  }
  then show ?case by auto
 qed
 thus ?thesis by simp
qed

lemma (in classical-logic) merge-witness-cons-measure-deduction:
 assumes map (uncurry (⊔)) Σ :⊢ φ
    and mset (map snd Δ) ⊆# mset (map (uncurry (→)) Σ @ Γ ⊖ map snd Σ)
    and map (uncurry (⊔)) Δ $⊢ Φ
  shows map (uncurry (⊔)) (𝔍 Σ Δ) $⊢ (φ # Φ)
proof −
 let ?Σ' = 𝔈 Σ Δ
 let ?Γ = map (uncurry (→)) ?Σ' @ map (uncurry (⊔)) (𝔍 Σ Δ) ⊖ map snd ?Σ'
 have ?Γ $⊢ Φ
   using assms(3)
        submerge-witness-stronger-theory-right
        measure-stronger-theory-left-monotonic
   by blast
 moreover have map (uncurry (⊔)) ?Σ' :⊢ φ
   using assms(1)
        stronger-theory-deduction-monotonic
        submerge-witness-stronger-theory-left
   by blast
 ultimately show ?thesis
   using submerge-witness-msub
   by fastforce
qed

primrec (in classical-logic)
 recover-witness-A :: ('a × 'a) list ⇒ ('a × 'a) list ⇒ ('a × 'a) list (𝔓)
 where
  𝔓 Σ [] = Σ
 | 𝔓 Σ (δ # Δ) =
     (case find (λ σ. snd σ = (uncurry (⊔)) δ) Σ of
          None ⇒ 𝔓 Σ Δ
        | Some σ ⇒ (fst σ ⊔ fst δ, snd δ) # (𝔓 (remove1 σ Σ) Δ))

primrec (in classical-logic)
 recover-complement-A :: ('a × 'a) list ⇒ ('a × 'a) list ⇒ ('a × 'a) list (𝔓^C)
 where
  𝔓^C Σ [] = []
```

```
| 𝔓^C Σ (δ # Δ) =
    (case find (λ σ. snd σ = (uncurry (⊔)) δ) Σ of
        None ⇒ δ # 𝔓^C Σ Δ
      | Some σ ⇒ (𝔓^C (remove1 σ Σ) Δ))

primrec (in classical-logic)
  recover-witness-B :: ('a × 'a) list ⇒ ('a × 'a) list ⇒ ('a × 'a) list (𝔘)
  where
    𝔘 Σ [] = []
  | 𝔘 Σ (δ # Δ) =
      (case find (λ σ. (snd σ) = (uncurry (⊔)) δ) Σ of
          None ⇒ δ # 𝔘 Σ Δ
        | Some σ ⇒ (fst δ, (fst σ ⊔ fst δ) → snd δ) # (𝔘 (remove1 σ Σ) Δ))

lemma (in classical-logic) recover-witness-A-left-stronger-theory:
  map (uncurry (⊔)) Σ ≼ map (uncurry (⊔)) (𝔓 Σ Δ)
proof −
  have ∀ Σ. map (uncurry (⊔)) Σ ≼ map (uncurry (⊔)) (𝔓 Σ Δ)
  proof (induct Δ)
    case Nil
    {
      fix Σ
      have map (uncurry (⊔)) Σ ≼ map (uncurry (⊔)) (𝔓 Σ [])
        by(induct Σ, simp+)
    }
    then show ?case by auto
  next
    case (Cons δ Δ)
    {
      fix Σ
      have map (uncurry (⊔)) Σ ≼ map (uncurry (⊔)) (𝔓 Σ (δ # Δ))
      proof (cases find (λ σ. snd σ = uncurry (⊔) δ) Σ = None)
        case True
        then show ?thesis using Cons by simp
      next
        case False
        from this obtain σ where
          σ: find (λσ. snd σ = uncurry (⊔) δ) Σ = Some σ
             snd σ = uncurry (⊔) δ
             σ ∈ set Σ
          using find-Some-predicate
                find-Some-set-membership
          by fastforce
        let ?α = fst σ
        let ?β = fst δ
        let ?γ = snd δ
        have uncurry (⊔) = (λδ. fst δ ⊔ snd δ) by fastforce
        hence ⊢ ((?α ⊔ ?β) ⊔ ?γ) → uncurry (⊔) σ
          using σ(2) biconditional-def disjunction-associativity
```

```
      by auto
    moreover
    have map (uncurry (⊔)) (remove1 σ Σ)
      ⪯ map (uncurry (⊔)) (𝔓 (remove1 σ Σ) Δ)
      using Cons by simp
    ultimately have map (uncurry (⊔)) (σ # (remove1 σ Σ))
                  ⪯ map (uncurry (⊔)) (𝔓 Σ (δ # Δ))
      using σ(1)
      by (simp, metis stronger-theory-left-right-cons)
    moreover
    from σ(3) have mset Σ = mset (σ # (remove1 σ Σ))
      by simp
     hence mset (map (uncurry (⊔)) Σ) = mset (map (uncurry (⊔)) (σ #
(remove1 σ Σ)))
      by (metis mset-map)
    hence map (uncurry (⊔)) Σ ⪯ map (uncurry (⊔)) (σ # (remove1 σ Σ))
      by (simp add: msub-stronger-theory-intro)
    ultimately show ?thesis
      using stronger-theory-transitive by blast
  qed
 }
 then show ?case by blast
qed
 thus ?thesis by auto
qed

lemma (in classical-logic) recover-witness-A-mset-equiv:
 assumes mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
 shows mset (map snd (𝔓 Σ Δ @ 𝔓^C Σ Δ)) = mset (map snd Δ)
proof −
 have ∀ Σ. mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
       ⟶ mset (map snd (𝔓 Σ Δ @ 𝔓^C Σ Δ)) = mset (map snd Δ)
 proof (induct Δ)
  case Nil
  then show ?case by simp
 next
  case (Cons δ Δ)
  {
   fix Σ :: ('a × 'a) list
   assume ⋆: mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) (δ # Δ))
   have mset (map snd (𝔓 Σ (δ # Δ) @ 𝔓^C Σ (δ # Δ))) = mset (map snd (δ
# Δ))
    proof (cases find (λ σ. snd σ = uncurry (⊔) δ) Σ = None)
     case True
     hence uncurry (⊔) δ ∉ set (map snd Σ)
     proof (induct Σ)
      case Nil
      then show ?case by simp
     next
```

```
        case (Cons σ Σ)
        then show ?case
          by (cases (uncurry (⊔)) δ = snd σ, fastforce+)
      qed
       moreover have mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ) +
{#uncurry (⊔) δ#}
        using ⋆ by fastforce
      ultimately have mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
        by (metis diff-single-trivial
                  in-multiset-in-set
                  subset-eq-diff-conv)
      then show ?thesis using Cons True by simp
    next
      case False
      from this obtain σ where
        σ: find (λσ. snd σ = uncurry (⊔) δ) Σ = Some σ
           snd σ = uncurry (⊔) δ
           σ ∈ set Σ
        using find-Some-predicate
              find-Some-set-membership
        by fastforce
      have A: mset (map snd Σ)
           ⊆# mset (map (uncurry (⊔)) Δ) + add-mset (uncurry (⊔) δ) (mset [])
        using ⋆ by auto
      have (fst σ, uncurry (⊔) δ) ∈# mset Σ
        by (metis (no-types) σ(2) σ(3) prod.collapse set-mset-mset)
      then have B: mset (map snd (remove1 (fst σ, uncurry (⊔) δ) Σ))
                 = mset (map snd Σ) − {#uncurry (⊔) δ#}
        by (meson remove1-pairs-list-projections-snd)
      have (fst σ, uncurry (⊔) δ) = σ
        by (metis σ(2) prod.collapse)
      then have mset (map snd Σ) − add-mset (uncurry (⊔) δ) (mset [])
            = mset (map snd (remove1 σ Σ))
        using B by simp
      hence mset (map snd (remove1 σ Σ)) ⊆# mset (map (uncurry (⊔)) Δ)
        using A by (metis (no-types) subset-eq-diff-conv)
      with σ(1) Cons show ?thesis by simp
    qed
  }
  then show ?case by simp
 qed
 with assms show ?thesis by blast
qed

lemma (in classical-logic) recover-witness-B-stronger-theory:
 assumes mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
 shows (map (uncurry (→)) Σ @ map (uncurry (⊔)) Δ ⊖ map snd Σ)
       ≼ map (uncurry (⊔)) (Ω Σ Δ)
proof −
```

```
  have ∀ Σ. mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
        ⟶ (map (uncurry (→)) Σ @ map (uncurry (⊔)) Δ ⊖ map snd Σ)
          ≼ map (uncurry (⊔)) (𝔘 Σ Δ)
  proof(induct Δ)
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    {
      fix Σ :: ('a × 'a) list
      assume ⋆: mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) (δ # Δ))
      have (map (uncurry (→)) Σ @ map (uncurry (⊔)) (δ # Δ) ⊖ map snd Σ)
          ≼ map (uncurry (⊔)) (𝔘 Σ (δ # Δ))
      proof (cases find (λ σ. snd σ = uncurry (⊔) δ) Σ = None)
        case True
        hence uncurry (⊔) δ ∉ set (map snd Σ)
        proof (induct Σ)
          case Nil
          then show ?case by simp
        next
          case (Cons σ Σ)
          then show ?case
            by (cases uncurry (⊔) δ = snd σ, fastforce+)
        qed
       hence mset (map (uncurry (→)) Σ @ (map (uncurry (⊔)) (δ # Δ)) ⊖ map
snd Σ)
            = mset (uncurry (⊔) δ # map (uncurry (→)) Σ
                  @ map (uncurry (⊔)) Δ ⊖ map snd Σ)
             mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
          using ⋆
          by (simp, simp,
              metis add-mset-add-single
                    diff-single-trivial
                    image-set
                    mset-map
                    set-mset-mset
                    subset-eq-diff-conv)
        moreover from this have
          (map (uncurry (→)) Σ @ map (uncurry (⊔)) Δ ⊖ map snd Σ)
           ≼ map (uncurry (⊔)) (𝔘 Σ Δ)
          using Cons
          by auto
        hence (uncurry (⊔) δ # map (uncurry (→)) Σ @ map (uncurry (⊔)) Δ ⊖
map snd Σ)
              ≼ map (uncurry (⊔)) (𝔘 Σ (δ # Δ))
          using True
          by (simp add: stronger-theory-left-right-cons trivial-implication)
        ultimately show ?thesis
          unfolding stronger-theory-relation-alt-def
```

```
      by simp
    next
     case False
    let ?Γ = map (uncurry (→)) Σ @ (map (uncurry (⊔)) (δ # Δ)) ⊖ map snd
Σ
     from False obtain σ where
      σ: find (λσ. snd σ = uncurry (⊔) δ) Σ = Some σ
         snd σ = uncurry (⊔) δ
         σ ∈ set Σ
      using find-Some-predicate
          find-Some-set-membership
      by fastforce
     let ?Γ₀ = map (uncurry (→)) (remove1 σ Σ)
               @ (map (uncurry (⊔)) Δ) ⊖ map snd (remove1 σ Σ)
     let ?α = fst σ
     let ?β = fst δ
     let ?γ = snd δ
     have uncurry (⊔) = (λ σ. fst σ ⊔ snd σ)
         uncurry (→) = (λ σ. fst σ → snd σ)
      by fastforce+
     hence uncurry (→) σ = ?α → (?β ⊔ ?γ)
      using σ(2)
      by simp
     from σ(3) have mset (σ # (remove1 σ Σ)) = mset Σ by simp
     hence ♠: mset (map snd (σ # (remove1 σ Σ))) = mset (map snd Σ)
               mset (map (uncurry (→)) (σ # (remove1 σ Σ))) = mset (map
(uncurry (→)) Σ)
      by (metis mset-map)+
     hence mset ?Γ = mset (map (uncurry (→)) (σ # (remove1 σ Σ))
                          @ (uncurry (⊔) δ # map (uncurry (⊔)) Δ)
                            ⊖ map snd (σ # (remove1 σ Σ)))
      by simp
     hence ?Γ ≼ (?α → (?β ⊔ ?γ) # ?Γ₀)
      using σ(2) ‹uncurry (→) σ = ?α → (?β ⊔ ?γ)›
      by (simp add: msub-stronger-theory-intro)
    moreover have mset (map snd (remove1 σ Σ)) ⊆# mset (map (uncurry
(⊔)) Δ)
      using ♠(1)
      by (simp,
          metis (no-types, lifting)
                ⋆ σ(2)
                list.simps(9)
                mset.simps(2)
                mset-map
                uncurry-def
                mset-subset-eq-add-mset-cancel)
    with Cons have ♡: ?Γ₀ ≼ map (uncurry (⊔)) (Ω (remove1 σ Σ) Δ) by
simp
     {
```

```
      fix α β γ
      have ⊢ (β ⊔ (α ⊔ β) → γ) → (α → (β ⊔ γ))
      proof −
        let ?φ = (⟨β⟩ ⊔ (⟨α⟩ ⊔ ⟨β⟩) → ⟨γ⟩) → (⟨α⟩ → (⟨β⟩ ⊔ ⟨γ⟩))
        have ∀ 𝔐. 𝔐 ⊨_prop ?φ by fastforce
        hence ⊢ ⦇ ?φ ⦈ using propositional-semantics by blast
        thus ?thesis by simp
      qed
    }
    hence ⊢ (?β ⊔ (?α ⊔ ?β) → ?γ) → (?α → (?β ⊔ ?γ))
      by simp
    hence (?α → (?β ⊔ ?γ) # ?Γ₀) ≼ map (uncurry (⊔)) (𝔘 Σ (δ # Δ))
      using σ(1) ♡
      by (simp, metis stronger-theory-left-right-cons)
    ultimately show ?thesis
      using stronger-theory-transitive by blast
  qed
 }
 then show ?case by simp
qed
thus ?thesis using assms by blast
qed

lemma (in classical-logic) recover-witness-B-mset-equiv:
 assumes mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
 shows mset (map snd (𝔘 Σ Δ))
     = mset (map (uncurry (→)) (𝔓 Σ Δ) @ map snd Δ ⊖ map snd (𝔓 Σ Δ))
proof −
 have ∀ Σ. mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
    ⟶ mset (map snd (𝔘 Σ Δ)) = mset (map (uncurry (→)) (𝔓 Σ Δ) @ map
snd (𝔓^C Σ Δ))
 proof (induct Δ)
  case Nil
  then show ?case by simp
 next
  case (Cons δ Δ)
  {
   fix Σ :: ('a × 'a) list
   assume ⋆: mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) (δ # Δ))
   have mset (map snd (𝔘 Σ (δ # Δ)))
     = mset (map (uncurry (→)) (𝔓 Σ (δ # Δ)) @ map snd (𝔓^C Σ (δ # Δ)))
   proof (cases find (λ σ. snd σ = uncurry (⊔) δ) Σ = None)
    case True
    hence uncurry (⊔) δ ∉ set (map snd Σ)
    proof (induct Σ)
     case Nil
     then show ?case by simp
    next
     case (Cons σ Σ)
```

```
        then show ?case
          by (cases (uncurry (⊔)) δ = snd σ, fastforce+)
      qed
       moreover have mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ) +
{#uncurry (⊔) δ#}
        using ⋆ by force
      ultimately have mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
        by (metis diff-single-trivial in-multiset-in-set subset-eq-diff-conv)
      then show ?thesis using True Cons by simp
    next
      case False
      from this obtain σ where
        σ: find (λσ. snd σ = uncurry (⊔) δ) Σ = Some σ
           snd σ = uncurry (⊔) δ
           σ ∈ set Σ
        using find-Some-predicate
              find-Some-set-membership
        by fastforce
      hence (fst σ, uncurry (⊔) δ) ∈# mset Σ
        by (metis (full-types) prod.collapse set-mset-mset)
      then have mset (map snd (remove1 (fst σ, uncurry (⊔) δ) Σ))
              = mset (map snd Σ) − {#uncurry (⊔) δ#}
        by (meson remove1-pairs-list-projections-snd)
      moreover have
      mset (map snd Σ)
    ⊆# mset (map (uncurry (⊔)) Δ) + add-mset (uncurry (⊔) δ) (mset [])
        using ⋆ by force
      ultimately have mset (map snd (remove1 σ Σ))
          ⊆# mset (map (uncurry (⊔)) Δ)
        by (metis (no-types) σ(2) mset.simps(1) prod.collapse subset-eq-diff-conv)
      with σ(1) Cons show ?thesis by simp
    qed
  }
  then show ?case by blast
 qed
 thus ?thesis
  using assms recover-witness-A-mset-equiv
  by (simp, metis add-diff-cancel-left′)
qed

lemma (in classical-logic) recover-witness-B-right-stronger-theory:
 map (uncurry (→)) Δ ≼ map (uncurry (→)) (𝔒 Σ Δ)
proof −
 have ∀ Σ. map (uncurry (→)) Δ ≼ map (uncurry (→)) (𝔒 Σ Δ)
 proof (induct Δ)
  case Nil
  then show ?case by simp
 next
  case (Cons δ Δ)
```

```
    {
     fix Σ
     have map (uncurry (→)) (δ # Δ) ≼ map (uncurry (→)) (𝔘 Σ (δ # Δ))
     proof (cases find (λ σ. snd σ = uncurry (⊔) δ) Σ = None)
      case True
      then show ?thesis
       using Cons
       by (simp add: stronger-theory-left-right-cons trivial-implication)
     next
      case False
      from this obtain σ where σ:
       find (λσ. snd σ = uncurry (⊔) δ) Σ = Some σ
       by fastforce
      let ?α = fst δ
      let ?β = snd δ
      let ?γ = fst σ
      have uncurry (→) = (λδ. fst δ → snd δ) by fastforce
      hence uncurry (→) δ = ?α → ?β by auto
      moreover have ⊢ (?α → (?γ ⊔ ?α) → ?β) → ?α → ?β
       unfolding disjunction-def
       using axiom-k axiom-s modus-ponens flip-implication
       by blast
      ultimately show ?thesis
       using Cons σ
       by (simp add: stronger-theory-left-right-cons)
     qed
    }
    then show ?case by simp
  qed
  thus ?thesis by simp
qed

lemma (in classical-logic) recoverWitnesses-mset-equiv:
  assumes mset (map snd Δ) ⊆# mset Γ
      and mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
    shows mset (Γ ⊖ map snd Δ)
        = mset ((map (uncurry (→)) (𝔓 Σ Δ) @ Γ ⊖ map snd (𝔓 Σ Δ)) ⊖ map
snd (𝔘 Σ Δ))
proof −
  have mset (Γ ⊖ map snd Δ) = mset (Γ ⊖ map snd (𝔓^C Σ Δ) ⊖ map snd (𝔓
Σ Δ))
    using assms(2) recover-witness-A-mset-equiv
    by (simp add: union-commute)
  moreover have ∀ Σ. mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
                ⟶ mset (Γ ⊖ map snd (𝔓^C Σ Δ))
                  = (mset ((map (uncurry (→)) (𝔓 Σ Δ) @ Γ) ⊖ map snd (𝔘 Σ
Δ)))
    using assms(1)
  proof (induct Δ)
```

```
    case Nil
    then show ?case by simp
  next
    case (Cons δ Δ)
    from Cons.prems have snd δ ∈ set Γ
      using mset-subset-eqD by fastforce
    from Cons.prems have ♡: mset (map snd Δ) ⊆# mset Γ
      using subset-mset.dual-order.trans
      by fastforce
    {
      fix Σ :: ('a × 'a) list
      assume ⋆: mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) (δ # Δ))
      have mset (Γ ⊖ map snd (𝔓^C Σ (δ # Δ)))
         = mset ((map (uncurry (→)) (𝔓 Σ (δ # Δ)) @ Γ) ⊖ map snd (𝔘 Σ (δ #
Δ)))
      proof (cases find (λ σ. snd σ = uncurry (⊔) δ) Σ = None)
        case True
        hence uncurry (⊔) δ ∉ set (map snd Σ)
        proof (induct Σ)
          case Nil
          then show ?case by simp
        next
          case (Cons σ Σ)
          then show ?case
            by (cases (uncurry (⊔)) δ = snd σ, fastforce+)
        qed
        moreover have mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ) +
{#uncurry (⊔) δ#}
          using ⋆ by auto
        ultimately have mset (map snd Σ) ⊆# mset (map (uncurry (⊔)) Δ)
        by (metis (full-types) diff-single-trivial in-multiset-in-set subset-eq-diff-conv)
        with Cons.hyps ♡ have mset (Γ ⊖ map snd (𝔓^C Σ Δ))
                            = mset ((map (uncurry (→)) (𝔓 Σ Δ) @ Γ) ⊖ map snd
(𝔘 Σ Δ))
          by simp
        thus ?thesis using True ‹snd δ ∈ set Γ› by simp
      next
        case False
        from this obtain σ where σ:
          find (λσ. snd σ = uncurry (⊔) δ) Σ = Some σ
          snd σ = uncurry (⊔) δ
          σ ∈ set Σ
          using find-Some-predicate
                find-Some-set-membership
          by fastforce
        with ⋆ have mset (map snd (remove1 σ Σ)) ⊆# mset (map (uncurry (⊔))
Δ)
          by (simp, metis (no-types, lifting)
                          add-mset-remove-trivial-eq
```

```
                    image-mset-add-mset
                    in-multiset-in-set
                    mset-subset-eq-add-mset-cancel)
      with Cons.hyps have mset (Γ ⊖ map snd (𝔓^C (remove1 σ Σ) Δ))
                        = mset ((map (uncurry (→)) (𝔓 (remove1 σ Σ) Δ) @ Γ)
                              ⊖ map snd (𝔘 (remove1 σ Σ) Δ))
        using ♡ by blast
      then show ?thesis using σ by simp
    qed
  }
  then show ?case by blast
 qed
 moreover have image-mset snd (mset (𝔓^C Σ Δ)) = mset (map snd Δ ⊖ map
snd (𝔓 Σ Δ))
  using assms(2) recover-witness-A-mset-equiv
  by (simp, metis (no-types) diff-union-cancelL list-subtract-mset-homomorphism
mset-map)
 then have mset Γ − (image-mset snd (mset (𝔓^C Σ Δ)) + image-mset snd (mset
(𝔓 Σ Δ)))
       = {#x → y. (x, y) ∈# mset (𝔓 Σ Δ)#}
         + (mset Γ − image-mset snd (mset (𝔓 Σ Δ))) − image-mset snd (mset
(𝔘 Σ Δ))
  using calculation
       assms(2)
       recover-witness-A-mset-equiv
       recover-witness-B-mset-equiv
  by fastforce
 ultimately
 show ?thesis
  using assms recover-witness-A-mset-equiv
  by simp
qed

theorem (in classical-logic) measure-deduction-generalized-witness:
 Γ $⊢ (Φ @ Ψ) = (∃ Σ. mset (map snd Σ) ⊆# mset Γ ∧
                   map (uncurry (⊔)) Σ $⊢ Φ ∧
                   (map (uncurry (→)) Σ @ Γ ⊖ (map snd Σ)) $⊢ Ψ)
proof −
 have ∀ Γ Ψ. Γ $⊢ (Φ @ Ψ) = (∃ Σ. mset (map snd Σ) ⊆# mset Γ ∧
                                map (uncurry (⊔)) Σ $⊢ Φ ∧
                                (map (uncurry (→)) Σ @ Γ ⊖ (map snd Σ)) $⊢ Ψ)
 proof (induct Φ)
  case Nil
  {
   fix Γ Ψ
   have Γ $⊢ ([] @ Ψ) = (∃Σ. mset (map snd Σ) ⊆# mset Γ ∧
                          map (uncurry (⊔)) Σ $⊢ [] ∧
                          map (uncurry (→)) Σ @ Γ ⊖ map snd Σ $⊢ Ψ)
   proof (rule iffI)
```

```
    assume Γ $⊢ ([] @ Ψ)
    moreover
    have Γ $⊢ ([] @ Ψ) = (mset (map snd []) ⊆# mset Γ ∧
                          map (uncurry (⊔)) [] $⊢ [] ∧
                          map (uncurry (→)) [] @ Γ ⊖ (map snd []) $⊢ Ψ)
      by simp
    ultimately show ∃Σ. mset (map snd Σ) ⊆# mset Γ ∧
                        map (uncurry (⊔)) Σ $⊢ [] ∧
                        map (uncurry (→)) Σ @ Γ ⊖ map snd Σ $⊢ Ψ
      by metis
  next
    assume ∃Σ. mset (map snd Σ) ⊆# mset Γ ∧
               map (uncurry (⊔)) Σ $⊢ [] ∧
               map (uncurry (→)) Σ @ Γ ⊖ map snd Σ $⊢ Ψ
    from this obtain Σ where
      Σ: mset (map snd Σ) ⊆# mset Γ
         map (uncurry (→)) Σ @ Γ ⊖ map snd Σ $⊢ ([] @ Ψ)
      by fastforce
    hence (map (uncurry (→)) Σ @ Γ ⊖ map snd Σ) ≼ Γ
      using witness-stronger-theory by auto
    with Σ(2) show Γ $⊢ ([] @ Ψ)
      using measure-stronger-theory-left-monotonic by blast
  qed
 }
 then show ?case by blast
next
 case (Cons φ Φ)
 {
  fix Γ Ψ
  have Γ $⊢ ((φ # Φ) @ Ψ) = (∃Σ. mset (map snd Σ) ⊆# mset Γ ∧
                              map (uncurry (⊔)) Σ $⊢ (φ # Φ) ∧
                              map (uncurry (→)) Σ @ Γ ⊖ map snd Σ $⊢ Ψ)
  proof (rule iffI)
    assume Γ $⊢ ((φ # Φ) @ Ψ)
    from this obtain Σ where
      Σ: mset (map snd Σ) ⊆# mset Γ
         map (uncurry (⊔)) Σ :⊢ φ
         map (uncurry (→)) Σ @ Γ ⊖ (map snd Σ) $⊢ (Φ @ Ψ)
         (is ?Γ₀ $⊢ (Φ @ Ψ))
      by auto
    from this(3) obtain Δ where
      Δ: mset (map snd Δ) ⊆# mset ?Γ₀
         map (uncurry (⊔)) Δ $⊢ Φ
         map (uncurry (→)) Δ @ ?Γ₀ ⊖ (map snd Δ) $⊢ Ψ
      using Cons
      by auto
    let ?Σ' = ℑ Σ Δ
    have map (uncurry (⊔)) ?Σ' $⊢ (φ # Φ)
      using Δ(1) Δ(2) Σ(2) merge-witness-cons-measure-deduction by blast
```

moreover have *mset* (*map snd* ?Σ′) ⊆# *mset* Γ
using Δ(*1*) Σ(*1*) *merge-witness-msub-intro* **by** *blast*
moreover have *map* (*uncurry* (→)) ?Σ′ @ Γ ⊖ *map snd* ?Σ′ $⊢ Ψ
using Δ(*1*) Δ(*3*) *merge-witness-measure-deduction-intro* **by** *blast*
ultimately show
∃Σ. *mset* (*map snd* Σ) ⊆# *mset* Γ ∧
map (*uncurry* (⊔)) Σ $⊢ (φ # Φ) ∧
map (*uncurry* (→)) Σ @ Γ ⊖ *map snd* Σ $⊢ Ψ
by *fast*
next
assume ∃Σ. *mset* (*map snd* Σ) ⊆# *mset* Γ ∧
map (*uncurry* (⊔)) Σ $⊢ (φ # Φ) ∧
map (*uncurry* (→)) Σ @ Γ ⊖ *map snd* Σ $⊢ Ψ
from *this* **obtain** Δ **where** Δ:
mset (*map snd* Δ) ⊆# *mset* Γ
map (*uncurry* (⊔)) Δ $⊢ (φ # Φ)
map (*uncurry* (→)) Δ @ Γ ⊖ *map snd* Δ $⊢ Ψ
by *auto*
from *this* **obtain** Σ **where** Σ:
mset (*map snd* Σ) ⊆# *mset* (*map* (*uncurry* (⊔)) Δ)
map (*uncurry* (⊔)) Σ :⊢ φ
map (*uncurry* (→)) Σ @ (*map* (*uncurry* (⊔)) Δ) ⊖ *map snd* Σ $⊢ Φ
by *auto*
let ?Ω = 𝔓 Σ Δ
let ?Ξ = 𝔔 Σ Δ
let ?Γ$_0$ = *map* (*uncurry* (→)) ?Ω @ Γ ⊖ *map snd* ?Ω
let ?Γ$_1$ = *map* (*uncurry* (→)) ?Ξ @ ?Γ$_0$ ⊖ *map snd* ?Ξ
have *mset* (Γ ⊖ *map snd* Δ) = *mset* (?Γ$_0$ ⊖ *map snd* ?Ξ)
using Δ(*1*) Σ(*1*) *recoverWitnesses-mset-equiv* **by** *blast*
hence (Γ ⊖ *map snd* Δ) ≼ (?Γ$_0$ ⊖ *map snd* ?Ξ)
by (*simp add*: *msub-stronger-theory-intro*)
hence ?Γ$_1$ $⊢ Ψ
using Δ(*3*) *measure-stronger-theory-left-monotonic*
stronger-theory-combine
recover-witness-B-right-stronger-theory
by *blast*
moreover
have *mset* (*map snd* ?Ξ) ⊆# *mset* ?Γ$_0$
using Σ(*1*) Δ(*1*) *recover-witness-B-mset-equiv*
by (*simp*,
metis list-subtract-monotonic
list-subtract-mset-homomorphism
mset-map)
moreover
have *map* (*uncurry* (⊔)) ?Ξ $⊢ Φ
using Σ(*1*) *recover-witness-B-stronger-theory*
Σ(*3*) *measure-stronger-theory-left-monotonic* **by** *blast*
ultimately have ?Γ$_0$ $⊢ (Φ @ Ψ)
using *Cons* **by** *fast*

```
      moreover
      have mset (map snd ?Ω) ⊆# mset (map snd Δ)
        using Σ(1) recover-witness-A-mset-equiv
        by (simp, metis mset-subset-eq-add-left)
      hence mset (map snd ?Ω) ⊆# mset Γ using Δ(1) by simp
      moreover
      have map (uncurry (⊔)) ?Ω :⊢ φ
        using Σ(2)
              recover-witness-A-left-stronger-theory
              stronger-theory-deduction-monotonic
        by blast
      ultimately show Γ $⊢ ((φ # Φ) @ Ψ)
        by (simp, blast)
    qed
  }
  then show ?case by metis
 qed
 thus ?thesis by blast
qed

lemma (in classical-logic) measure-list-deduction-antitonic:
 assumes Γ $⊢ Ψ
    and Ψ :⊢ φ
  shows Γ :⊢ φ
 using assms
proof (induct Ψ arbitrary: Γ φ)
 case Nil
 then show ?case
  using list-deduction-weaken
  by simp
next
 case (Cons ψ Ψ)
 hence Ψ :⊢ ψ → φ
  using list-deduction-theorem by blast
 from ‹Γ $⊢ (ψ # Ψ)› obtain Σ where Σ:
  mset (map snd Σ) ⊆# mset Γ
  map (uncurry (⊔)) Σ :⊢ ψ
  map (uncurry (→)) Σ @ Γ ⊖ map snd Σ $⊢ Ψ
  by auto
 hence Γ :⊢ ψ → φ
  using
    measure-stronger-theory-left-monotonic
    witness-stronger-theory
    ‹Ψ :⊢ ψ → φ›
    Cons
  by blast
 moreover
 have Γ :⊢ ψ
  using Σ(1) Σ(2)
```

stronger-theory-deduction-monotonic
witness-weaker-theory
by *blast*
ultimately show *?case* **using** *list-deduction-modus-ponens* **by** *auto*
qed

Finally, we may establish that ($\$\vdash$) is transitive.

theorem (**in** *classical-logic*) *measure-transitive*:
assumes Γ $\$\vdash$ Λ
and Λ $\$\vdash$ Δ
shows Γ $\$\vdash$ Δ
using *assms*
proof (*induct* Δ *arbitrary*: Γ Λ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* δ Δ)
from *this* **obtain** Σ **where** Σ:
mset (*map snd* Σ) $\subseteq\#$ *mset* Λ
map (*uncurry* ($\sqcup$)) Σ :$\vdash$ δ
map (*uncurry* ($\rightarrow$)) Σ @ Λ $\ominus$ *map snd* Σ $\$\vdash$ Δ
by *auto*
hence Γ $\$\vdash$ (*map* (*uncurry* ($\sqcup$)) Σ @ *map* (*uncurry* ($\rightarrow$)) Σ @ Λ $\ominus$ (*map snd* Σ))
using *Cons measure-witness-right-split*
by *simp*
from *this* **obtain** Ψ **where** Ψ:
mset (*map snd* Ψ) $\subseteq\#$ *mset* Γ
map (*uncurry* ($\sqcup$)) Ψ $\$\vdash$ *map* (*uncurry* ($\sqcup$)) Σ
map (*uncurry* ($\rightarrow$)) Ψ @ Γ $\ominus$ *map snd* Ψ $\$\vdash$ (*map* (*uncurry* ($\rightarrow$)) Σ @ Λ $\ominus$ *map snd* Σ)
using *measure-deduction-generalized-witness*
by *fastforce*
have *map* (*uncurry* ($\rightarrow$)) Ψ @ Γ $\ominus$ *map snd* Ψ $\$\vdash$ Δ
using $\Sigma(3)$ $\Psi(3)$ *Cons*
by *auto*
moreover
have *map* (*uncurry* ($\sqcup$)) Ψ :$\vdash$ δ
using $\Psi(2)$ $\Sigma(2)$ *measure-list-deduction-antitonic*
by *blast*
ultimately show *?case*
using $\Psi(1)$
by *fastforce*
qed

2.6 Measure Deduction Cancellation Rules

In this chapter we go over how to cancel formulae occurring in measure deduction judgements.

The first observation is that tautologies can always be canceled on either side of the turnstile.

lemma (**in** *classical-logic*) *measure-tautology-right-cancel*:
  assumes $\vdash \varphi$
  shows $\Gamma \$\vdash (\varphi \# \Phi) = \Gamma \$\vdash \Phi$
proof (*rule iffI*)
  assume $\Gamma \$\vdash (\varphi \# \Phi)$
  from *this* **obtain** Σ **where** Σ:
    $mset\ (map\ snd\ \Sigma) \subseteq\# mset\ \Gamma$
    $map\ (uncurry\ (\sqcup))\ \Sigma :\vdash \varphi$
    $map\ (uncurry\ (\rightarrow))\ \Sigma\ @\ \Gamma \ominus map\ snd\ \Sigma \$\vdash \Phi$
    by *auto*
  thus $\Gamma \$\vdash \Phi$
    using *measure-stronger-theory-left-monotonic*
        witness-stronger-theory
    by *blast*
next
  assume $\Gamma \$\vdash \Phi$
  hence $map\ (uncurry\ (\rightarrow))\ []\ @\ \Gamma \ominus map\ snd\ []\ \$\vdash \Phi$
      $mset\ (map\ snd\ []) \subseteq\# mset\ \Gamma$
      $map\ (uncurry\ (\sqcup))\ [] :\vdash \varphi$
    using *assms*
    by *simp+*
  thus $\Gamma \$\vdash (\varphi \# \Phi)$
    using *measure-deduction.simps*(*2*)
    by *blast*
qed

lemma (**in** *classical-logic*) *measure-tautology-left-cancel* [*simp*]:
  assumes $\vdash \gamma$
  shows $(\gamma \# \Gamma) \$\vdash \Phi = \Gamma \$\vdash \Phi$
proof (*rule iffI*)
  assume $(\gamma \# \Gamma) \$\vdash \Phi$
  moreover have $\Gamma \$\vdash \Gamma$
    by (*simp add*: *stronger-theory-to-measure-deduction*)
  hence $\Gamma \$\vdash (\gamma \# \Gamma)$
    using *assms measure-tautology-right-cancel*
    by *simp*
  ultimately show $\Gamma \$\vdash \Phi$
    using *measure-transitive* **by** *blast*
next
  assume $\Gamma \$\vdash \Phi$
  moreover have $mset\ \Gamma \subseteq\# mset\ (\gamma \# \Gamma)$
    by *simp*

```
  hence (γ # Γ) $⊢ Γ
    using msub-stronger-theory-intro
          stronger-theory-to-measure-deduction
    by blast
  ultimately show (γ # Γ) $⊢ Φ
    using measure-transitive by blast
qed


lemma (in classical-logic) measure-deduction-one-collapse:
  Γ $⊢ [φ] = Γ :⊢ φ
proof (rule iffI)
  assume Γ $⊢ [φ]
  from this obtain Σ where
    Σ: mset (map snd Σ) ⊆# mset Γ
       map (uncurry (⊔)) Σ :⊢ φ
    by auto
  hence map (uncurry (⊔)) Σ ≼ Γ
    using witness-weaker-theory by blast
  thus Γ :⊢ φ using Σ(2)
    using stronger-theory-deduction-monotonic by blast
next
  assume Γ :⊢ φ
  let ?Σ = map (λ γ. (⊥, γ)) Γ
  have Γ ≼ map (uncurry (⊔)) ?Σ
  proof (induct Γ)
    case Nil
    then show ?case by simp
  next
    case (Cons γ Γ)
    have ⊢ (⊥ ⊔ γ) → γ
      unfolding disjunction-def
      using ex-falso-quodlibet modus-ponens excluded-middle-elimination
      by blast
    then show ?case using Cons
      by (simp add: stronger-theory-left-right-cons)
  qed
  hence map (uncurry (⊔)) ?Σ :⊢ φ
    using ‹Γ :⊢ φ› stronger-theory-deduction-monotonic by blast
  moreover have mset (map snd ?Σ) ⊆# mset Γ by (induct Γ, simp+)
  ultimately show Γ $⊢ [φ]
    using measure-deduction.simps(1)
          measure-deduction.simps(2)
    by blast
qed
```

Split cases, which are occurrences of $\psi \sqcup \varphi \# \psi \to \varphi \# \ldots$, also cancel and simplify to just $\varphi \# \ldots$. We previously established $\Gamma \$\vdash \psi \sqcup \varphi \# \psi \to \varphi \# \Phi = \Gamma \$\vdash \varphi \# \Phi$ as part of the proof of transitivity.

lemma (**in** *classical-logic*) *measure-formula-left-split*:
  $\psi \sqcup \varphi \# \psi \rightarrow \varphi \# \Gamma \$\vdash \Phi = \varphi \# \Gamma \$\vdash \Phi$
proof (*rule iffI*)
  assume $\varphi \# \Gamma \$\vdash \Phi$
  have $\psi \sqcup \varphi \# \psi \rightarrow \varphi \# \Gamma \$\vdash (\psi \sqcup \varphi \# \psi \rightarrow \varphi \# \Gamma)$
    using *stronger-theory-to-measure-deduction*
        stronger-theory-reflexive
    by *blast*
  hence $\psi \sqcup \varphi \# \psi \rightarrow \varphi \# \Gamma \$\vdash (\varphi \# \Gamma)$
    using *measure-formula-right-split* **by** *blast*
  with ‹$\varphi \# \Gamma \$\vdash \Phi$› **show** $\psi \sqcup \varphi \# \psi \rightarrow \varphi \# \Gamma \$\vdash \Phi$
    using *measure-transitive* **by** *blast*
next
  assume $\psi \sqcup \varphi \# \psi \rightarrow \varphi \# \Gamma \$\vdash \Phi$
  have $\varphi \# \Gamma \$\vdash (\varphi \# \Gamma)$
    using *stronger-theory-to-measure-deduction*
        stronger-theory-reflexive
    by *blast*
  hence $\varphi \# \Gamma \$\vdash (\psi \sqcup \varphi \# \psi \rightarrow \varphi \# \Gamma)$
    using *measure-formula-right-split* **by** *blast*
  with ‹$\psi \sqcup \varphi \# \psi \rightarrow \varphi \# \Gamma \$\vdash \Phi$› **show** $\varphi \# \Gamma \$\vdash \Phi$
    using *measure-transitive* **by** *blast*
qed

lemma (**in** *classical-logic*) *measure-witness-left-split* [*simp*]:
  assumes *mset* (*map snd* Σ) $\subseteq\#$ *mset* Γ
  shows (*map* (*uncurry* ($\sqcup$)) Σ @ *map* (*uncurry* ($\rightarrow$)) Σ @ $\Gamma \ominus$ (*map snd* Σ)) $\$\vdash$ $\Phi = \Gamma \$\vdash \Phi$
  using *assms*
proof (*induct* Σ *arbitrary*: Γ)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons* σ Σ)
  let *?*χ = *fst* σ
  let *?*γ = *snd* σ
  let *?*Γ_0 = *map* (*uncurry* ($\sqcup$)) Σ @ *map* (*uncurry* ($\rightarrow$)) Σ @ $\Gamma \ominus$ *map snd* (σ # Σ)
  let *?*Γ' = *map* (*uncurry* ($\sqcup$)) (σ # Σ) @ *map* (*uncurry* ($\rightarrow$)) (σ # Σ) @ $\Gamma \ominus$ *map snd* (σ # Σ)
  assume *mset* (*map snd* (σ # Σ)) $\subseteq\#$ *mset* Γ
  hence *A*: *add-mset* (*snd* σ) (*image-mset snd* (*mset* Σ)) $\subseteq\#$ *mset* Γ **by** *simp*
  hence *B*: *image-mset snd* (*mset* Σ) + (*mset* Γ − *image-mset snd* (*mset* Σ))
      = *add-mset* (*snd* σ) (*image-mset snd* (*mset* Σ))
        + (*mset* Γ − *add-mset* (*snd* σ) (*image-mset snd* (*mset* Σ)))
    by (*metis* (*no-types*) *mset-subset-eq-insertD subset-mset.add-diff-inverse subset-mset-def*)
  have {#$x \rightarrow y$. (x, y) $\in\#$ *mset* Σ#}
        + *mset* Γ − *add-mset* (*snd* σ) (*image-mset snd* (*mset* Σ))

$= \{\#x \to y.\ (x,\ y) \in\# \ mset\ \Sigma\#\}$
$+ (mset\ \Gamma - add\text{-}mset\ (snd\ \sigma)\ (image\text{-}mset\ snd\ (mset\ \Sigma)))$
using *A subset-mset.diff-add-assoc* **by** *blast*
hence $\{\#x \to y.\ (x,\ y) \in\# \ mset\ \Sigma\#\} + (mset\ \Gamma - image\text{-}mset\ snd\ (mset\ \Sigma))$
$= add\text{-}mset\ (snd\ \sigma)\ (\{\#x \to y.\ (x,\ y) \in\# \ mset\ \Sigma\#\}$
$+ mset\ \Gamma - add\text{-}mset\ (snd\ \sigma)\ (image\text{-}mset\ snd\ (mset\ \Sigma)))$
using *B* **by** *auto*
hence *C*:
$mset\ (map\ snd\ \Sigma) \subseteq\# \ mset\ \Gamma$
$mset\ (map\ (uncurry\ (\sqcup))\ \Sigma\ @\ map\ (uncurry\ (\to))\ \Sigma\ @\ \Gamma \ominus map\ snd\ \Sigma)$
$= mset\ (?\gamma\ \#\ ?\Gamma_0)$
using ‹$mset\ (map\ snd\ (\sigma\ \#\ \Sigma)) \subseteq\# \ mset\ \Gamma$›
subset-mset.dual-order.trans
by (*fastforce+*)
hence $\Gamma \ \$\vdash \Phi = (?\chi \sqcup ?\gamma\ \#\ ?\chi \to ?\gamma\ \#\ ?\Gamma_0) \ \$\vdash \Phi$
proof −
have $\forall \Gamma\ \Delta.\ \neg\ mset\ (map\ snd\ \Sigma) \subseteq\# \ mset\ \Gamma$
$\vee\ \neg\ \Gamma \ \$\vdash \Phi$
$\vee\ \neg\ mset\ (map\ (uncurry\ (\sqcup))\ \Sigma$
$@\ map\ (uncurry\ (\to))\ \Sigma$
$@\ \Gamma \ominus map\ snd\ \Sigma)$
$\subseteq\# \ mset\ \Delta$
$\vee\ \Delta \ \$\vdash \Phi$
using *Cons.hyps measure-msub-left-monotonic* **by** *blast*
moreover
{
assume $\neg\ \Gamma \ \$\vdash \Phi$
then have $\exists \Delta.\ mset\ (snd\ \sigma\ \#\ map\ (uncurry\ (\sqcup))\ \Sigma$
$@\ map\ (uncurry\ (\to))\ \Sigma$
$@\ \Gamma \ominus map\ snd\ (\sigma\ \#\ \Sigma))$
$\subseteq\# \ mset\ \Delta$
$\wedge\ \neg\ \Gamma \ \$\vdash \Phi$
$\wedge\ \neg\ \Delta \ \$\vdash \Phi$
by (*metis* (*no-types*) *Cons.hyps C subset-mset.dual-order.refl*)
then have *?thesis*
using *measure-formula-left-split measure-msub-left-monotonic* **by** *blast*
}
ultimately show *?thesis*
by (*metis* (*full-types*) *C measure-formula-left-split subset-mset.dual-order.refl*)
qed
moreover
have $(uncurry\ (\sqcup)) = (\lambda\ \psi.\ fst\ \psi \sqcup snd\ \psi)$
$(uncurry\ (\to)) = (\lambda\ \psi.\ fst\ \psi \to snd\ \psi)$
by *fastforce+*
hence $mset\ ?\Gamma' = mset\ (?\chi \sqcup ?\gamma\ \#\ ?\chi \to ?\gamma\ \#\ ?\Gamma_0)$
by *fastforce*
hence $(?\chi \sqcup ?\gamma\ \#\ ?\chi \to ?\gamma\ \#\ ?\Gamma_0) \ \$\vdash \Phi = ?\Gamma' \ \$\vdash \Phi$
by (*metis*
(*mono-tags*, *lifting*)

```
        measure-msub-left-monotonic
        subset-mset.dual-order.refl)
  ultimately have Γ $⊢ Φ = ?Γ' $⊢ Φ
    by fastforce
  then show ?case by blast
qed
```

We now have enough to establish the cancellation rule for ($⊢).

```
lemma (in classical-logic) measure-cancel: (Δ @ Γ) $⊢ (Δ @ Φ) = Γ $⊢ Φ
proof −
  {
    fix Δ Γ Φ
    assume Γ $⊢ Φ
    hence (Δ @ Γ) $⊢ (Δ @ Φ)
    proof (induct Δ)
      case Nil
      then show ?case by simp
    next
      case (Cons δ Δ)
      let ?Σ = [(δ, δ)]
      have map (uncurry (⊔)) ?Σ :⊢ δ
        unfolding disjunction-def list-deduction-def
        by (simp add: Peirces-law)
      moreover have mset (map snd ?Σ) ⊆# mset (δ # Δ) by simp
      moreover have map (uncurry (→)) ?Σ @ ((δ # Δ) @ Γ) ⊖ map snd ?Σ $⊢
(Δ @ Φ)
        using Cons
        by (simp add: trivial-implication)
      moreover have map snd [(δ, δ)] = [δ] by force
      ultimately show ?case
        by (metis (no-types) measure-deduction.simps(2)
                             append-Cons
                             list.set-intros(1)
                             mset.simps(1)
                             mset.simps(2)
                             mset-subset-eq-single
                             set-mset-mset)
    qed
  } note forward-direction = this
  {
    assume (Δ @ Γ) $⊢ (Δ @ Φ)
    hence Γ $⊢ Φ
    proof (induct Δ)
      case Nil
      then show ?case by simp
    next
      case (Cons δ Δ)
      have mset ((δ # Δ) @ Φ) = mset ((Δ @ Φ) @ [δ]) by simp
      with Cons.prems have ((δ # Δ) @ Γ) $⊢ ((Δ @ Φ) @ [δ])
```

```
    by (metis measure-msub-weaken
          subset-mset.dual-order.refl)
  from this obtain Σ where Σ:
    mset (map snd Σ) ⊆# mset ((δ # Δ) @ Γ)
    map (uncurry (⊔)) Σ $⊢ (Δ @ Φ)
    map (uncurry (→)) Σ @ ((δ # Δ) @ Γ) ⊖ map snd Σ $⊢ [δ]
    by (metis append-assoc measure-deduction-generalized-witness)
  show ?case
  proof (cases find (λ σ. snd σ = δ) Σ = None)
    case True
    hence δ ∉ set (map snd Σ)
    proof (induct Σ)
      case Nil
      then show ?case by simp
    next
      case (Cons σ Σ)
      then show ?case by (cases snd σ = δ, simp+)
    qed
    with Σ(1) have mset (map snd Σ) ⊆# mset (Δ @ Γ)
      by (simp, metis add-mset-add-single
                      diff-single-trivial
                      mset-map
                      set-mset-mset
                      subset-eq-diff-conv)
    thus ?thesis
      using measure-stronger-theory-left-monotonic
          witness-weaker-theory
          Cons.hyps Σ(2)
      by blast
  next
    case False
    from this obtain σ χ where
      σ: σ = (χ, δ)
        σ ∈ set Σ
      using find-Some-predicate
          find-Some-set-membership
      by fastforce
    let ?Σ′ = remove1 σ Σ
    let ?Σ_A = map (uncurry (⊔)) ?Σ′
    let ?Σ_B = map (uncurry (→)) ?Σ′
    have mset Σ = mset (?Σ′ @ [(χ, δ)])
        mset Σ = mset ((χ, δ) # ?Σ′)
      using σ by simp+
   hence mset (map (uncurry (⊔)) Σ) = mset (map (uncurry (⊔)) (?Σ′ @ [(χ,
δ)]))
        mset (map snd Σ) = mset (map snd ((χ, δ) # ?Σ′))
         mset (map (uncurry (→)) Σ) = mset (map (uncurry (→)) ((χ, δ) #
?Σ′))
      by (metis mset-map)+
```

hence *mset* (*map* (*uncurry* (⊔)) Σ) = *mset* (*?Σ*$_A$ @ [χ ⊔ δ])
mset (*map* (*uncurry* (→)) Σ @ ((δ # Δ) @ Γ) ⊖ *map snd* Σ)
= *mset* (χ → δ # *?Σ*$_B$ @ (Δ @ Γ) ⊖ *map snd ?Σ′*)
by *simp+*
hence
?Σ$_A$ @ [χ ⊔ δ] $⊢ (Δ @ Φ)
χ → δ # (*?Σ*$_B$ @ (Δ @ Γ) ⊖ *map snd ?Σ′*) $⊢ [δ]
using Σ(*2*) Σ(*3*)
by (*metis measure-msub-left-monotonic subset-mset.dual-order.refl*, *simp*)
moreover
have ⊢ ((χ → δ) → δ) → (χ ⊔ δ)
unfolding *disjunction-def*
using *modus-ponens*
pseudo-scotus
flip-hypothetical-syllogism
by *blast*
ultimately have (*?Σ*$_A$ @ *?Σ*$_B$ @ (Δ @ Γ) ⊖ *map snd ?Σ′*) $⊢ (Δ @ Φ)
using *measure-deduction-one-collapse*
list-deduction-theorem
list-deduction-modus-ponens
list-deduction-weaken
forward-direction
measure-transitive
by *meson*
moreover
have δ = *snd* σ
snd σ ∈ *set* (*map snd* Σ)
by (*simp add*: σ(*1*), *simp add*: σ(*2*))
with Σ(*1*) **have** *mset* (*map snd* (*remove1* σ Σ)) ⊆# *mset* (*remove1* δ ((δ # Δ) @ Γ))
by (*metis insert-DiffM*
insert-subset-eq-iff
mset-remove1
σ(*1*) σ(*2*)
remove1-pairs-list-projections-snd
set-mset-mset)
hence *mset* (*map snd* (*remove1* σ Σ)) ⊆# *mset* (Δ @ Γ) **by** *simp*
ultimately show *?thesis*
using *measure-witness-left-split Cons.hyps*
by *blast*
qed
qed
}
with *forward-direction* **show** *?thesis* **by** *auto*
qed

lemma (**in** *classical-logic*) *measure-biconditional-cancel*:
assumes ⊢ γ ↔ φ
shows (γ # Γ) $⊢ (φ # Φ) = Γ $⊢ Φ

proof −
from *assms* **have** $(\gamma \# \Phi) \preceq (\varphi \# \Phi)$ $(\varphi \# \Phi) \preceq (\gamma \# \Phi)$
unfolding *biconditional-def*
by (*simp add*: *stronger-theory-left-right-cons*)+
hence $(\gamma \# \Phi)$ \$⊢ $(\varphi \# \Phi)$
$(\varphi \# \Phi)$ \$⊢ $(\gamma \# \Phi)$
using *stronger-theory-to-measure-deduction* **by** *blast*+
moreover
have Γ \$⊢ $\Phi = (\gamma \# \Gamma)$ \$⊢ $(\gamma \# \Phi)$
by (*metis append-Cons append-Nil measure-cancel*)+
ultimately
have Γ \$⊢ $\Phi \Longrightarrow \gamma \# \Gamma$ \$⊢ $(\varphi \# \Phi)$
$\gamma \# \Gamma$ \$⊢ $(\varphi \# \Phi) \Longrightarrow \Gamma$ \$⊢ Φ
using *measure-transitive* **by** *blast*+
thus *?thesis* **by** *blast*
qed

2.7 Measure Deduction Substitution Rules

Just like conventional deduction, if two formulae are equivalent then they may be substituted for one another.

lemma (**in** *classical-logic*) *right-measure-sub*:
assumes $\vdash \varphi \leftrightarrow \psi$
shows Γ \$⊢ $(\varphi \# \Phi) = \Gamma$ \$⊢ $(\psi \# \Phi)$
proof −
have Γ \$⊢ $(\varphi \# \Phi) = (\psi \# \Gamma)$ \$⊢ $(\psi \# \varphi \# \Phi)$
using *measure-cancel* [**where** Δ=[ψ] **and** Γ=Γ **and** Φ=$\varphi \# \Phi$] **by** *simp*
also have ... = $(\psi \# \Gamma)$ \$⊢ $(\varphi \# \psi \# \Phi)$
using *measure-cons-cons-right-permute* **by** *blast*
also have ... = Γ \$⊢ $(\psi \# \Phi)$
using *assms biconditional-symmetry-rule measure-biconditional-cancel* **by** *blast*
finally show *?thesis* **.**
qed

lemma (**in** *classical-logic*) *left-measure-sub*:
assumes $\vdash \gamma \leftrightarrow \chi$
shows $(\gamma \# \Gamma)$ \$⊢ $\Phi = (\chi \# \Gamma)$ \$⊢ Φ
proof −
have $(\gamma \# \Gamma)$ \$⊢ $\Phi = (\chi \# \gamma \# \Gamma)$ \$⊢ $(\chi \# \Phi)$
using *measure-cancel* [**where** Δ=[χ] **and** Γ=$(\gamma \# \Gamma)$ **and** Φ=Φ] **by** *simp*
also have ... = $(\gamma \# \chi \# \Gamma)$ \$⊢ $(\chi \# \Phi)$
using
measure-cons-cons-right-permute
stronger-theory-to-measure-deduction
measure-transitive
stronger-theory-reflexive
by *blast*
also have ... = $(\chi \# \Gamma)$ \$⊢ Φ

using *assms biconditional-symmetry-rule measure-biconditional-cancel* **by** *blast*
finally show *?thesis* **.**
qed

2.8 Measure Deduction Sum Rules

We next establish analogues of the rule in probability that $\mathcal{P}\ \alpha + \mathcal{P}\ \beta = \mathcal{P}\ (\alpha \sqcup \beta) + \mathcal{P}\ (\alpha \sqcap \beta)$. This equivalence holds for both sides of the ($\$\vdash$) turnstile.

lemma (**in** *classical-logic*) *right-measure-sum-rule*:
$\Gamma\ \$\vdash\ (\alpha\ \#\ \beta\ \#\ \Phi) = \Gamma\ \$\vdash\ (\alpha \sqcup \beta\ \#\ \alpha \sqcap \beta\ \#\ \Phi)$
proof −
have *A*: *mset* $(\alpha \sqcup \beta\ \#\ \beta \to \alpha\ \#\ \beta\ \#\ \Phi) = mset\ (\beta \to \alpha\ \#\ \beta\ \#\ \alpha \sqcup \beta\ \#\ \Phi)$
by *simp*
have *B*: $\vdash\ (\beta \to \alpha) \leftrightarrow (\beta \to (\alpha \sqcap \beta))$
proof −
let *?φ* $= (\langle\beta\rangle \to \langle\alpha\rangle) \leftrightarrow (\langle\beta\rangle \to (\langle\alpha\rangle \sqcap \langle\beta\rangle))$
have $\forall\,\mathfrak{M}.\ \mathfrak{M} \models_{prop}$ *?φ* **by** *fastforce*
hence $\vdash$ ⦇ *?φ* ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
have *C*: $\vdash\ \beta \leftrightarrow (\beta \sqcup (\alpha \sqcap \beta))$
proof −
let *?φ* $= \langle\beta\rangle \leftrightarrow (\langle\beta\rangle \sqcup (\langle\alpha\rangle \sqcap \langle\beta\rangle))$
have $\forall\,\mathfrak{M}.\ \mathfrak{M} \models_{prop}$ *?φ* **by** *fastforce*
hence $\vdash$ ⦇ *?φ* ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
have $\Gamma\ \$\vdash\ (\alpha\ \#\ \beta\ \#\ \Phi) = \Gamma\ \$\vdash\ (\beta \sqcup \alpha\ \#\ \beta \to \alpha\ \#\ \beta\ \#\ \Phi)$
using *measure-formula-right-split* **by** *blast*
also have ... $= \Gamma\ \$\vdash\ (\alpha \sqcup \beta\ \#\ \beta \to \alpha\ \#\ \beta\ \#\ \Phi)$
using *disjunction-commutativity right-measure-sub* **by** *blast*
also have ... $= \Gamma\ \$\vdash\ (\beta \to \alpha\ \#\ \beta\ \#\ \alpha \sqcup \beta\ \#\ \Phi)$
by (*metis A measure-msub-weaken subset-mset.dual-order.refl*)
also have ... $= \Gamma\ \$\vdash\ (\beta \to (\alpha \sqcap \beta)\ \#\ \beta\ \#\ \alpha \sqcup \beta\ \#\ \Phi)$
using *B right-measure-sub* **by** *blast*
also have ... $= \Gamma\ \$\vdash\ (\beta\ \#\ \beta \to (\alpha \sqcap \beta)\ \#\ \alpha \sqcup \beta\ \#\ \Phi)$
using *measure-cons-cons-right-permute* **by** *blast*
also have ... $= \Gamma\ \$\vdash\ (\beta \sqcup (\alpha \sqcap \beta)\ \#\ \beta \to (\alpha \sqcap \beta)\ \#\ \alpha \sqcup \beta\ \#\ \Phi)$
using *C right-measure-sub* **by** *blast*
also have ... $= \Gamma\ \$\vdash\ (\alpha \sqcap \beta\ \#\ \alpha \sqcup \beta\ \#\ \Phi)$
using *measure-formula-right-split* **by** *blast*
finally show *?thesis*
using *measure-cons-cons-right-permute* **by** *blast*
qed

lemma (**in** *classical-logic*) *left-measure-sum-rule*:
$(\alpha\ \#\ \beta\ \#\ \Gamma)\ \$\vdash\ \Phi = (\alpha \sqcup \beta\ \#\ \alpha \sqcap \beta\ \#\ \Gamma)\ \$\vdash\ \Phi$

proof −
have ⋆: *mset* (α ⊔ β # α ⊓ β # α # β # Γ) = *mset* (α # β # α ⊔ β # α ⊓ β # Γ) **by** *simp*
have (α # β # Γ) $⊢ Φ = (α ⊔ β # α ⊓ β # α # β # Γ) $⊢ (α ⊔ β # α ⊓ β # Φ)
using *measure-cancel* [**where** Δ=[α ⊔ β, α ⊓ β] **and** Γ=(α # β # Γ) **and** Φ=Φ] **by** *simp*
also have ... = (α ⊔ β # α ⊓ β # α # β # Γ) $⊢ (α # β # Φ)
using *right-measure-sum-rule* **by** *blast*
also have ... = (α # β # α ⊔ β # α ⊓ β # Γ) $⊢ (α # β # Φ)
by (*metis* ⋆ *measure-msub-left-monotonic subset-mset.dual-order.refl*)
also have ... = (α ⊔ β # α ⊓ β # Γ) $⊢ Φ
using *measure-cancel* [**where** Δ=[α, β] **and** Γ=(α ⊔ β # α ⊓ β # Γ) **and** Φ=Φ] **by** *simp*
finally show *?thesis* **.**
qed

2.9 Measure Deduction Exchange Rule

As we will see, a key result is that we can move formulae from the right hand side of the ($⊢) turnstile to the left.

We observe a novel logical principle, which we call *exchange*. This principle follows immediately from the split rules and cancellation rules.

lemma (**in** *classical-logic*) *measure-exchange*:
(γ # Γ) $⊢ (φ # Φ) = (φ → γ # Γ) $⊢ (γ → φ # Φ)
proof −
have (γ # Γ) $⊢ (φ # Φ) = (φ ⊔ γ # φ → γ # Γ) $⊢ (γ ⊔ φ # γ → φ # Φ)
using *measure-formula-left-split*
measure-formula-right-split
by *blast*+
thus *?thesis*
using *measure-biconditional-cancel*
disjunction-commutativity
by *blast*
qed

The exchange rule allows us to prove an analogue of the rule in classical logic that Γ :⊢ φ = (∼ φ # Γ) :⊢ ⊥ for measure deduction.

theorem (**in** *classical-logic*) *measure-negation-swap*:
Γ $⊢ (φ # Φ) = (∼ φ # Γ) $⊢ (⊥ # Φ)
proof −
have Γ $⊢ (φ # Φ) = (⊥ # Γ) $⊢ (⊥ # φ # Φ)
by (*metis append-Cons append-Nil measure-cancel*)
also have ... = (⊥ # Γ) $⊢ (φ # ⊥ # Φ)
using *measure-cons-cons-right-permute* **by** *blast*
also have ... = (∼ φ # Γ) $⊢ (⊥ → φ # ⊥ # Φ)
unfolding *negation-def*

using *measure-exchange*
by *blast*
also have ... = ($\sim \varphi$ # Γ) $\$\vdash$ ($\bot$ # Φ)
using *ex-falso-quodlibet*
measure-tautology-right-cancel
by *blast*
finally show *?thesis* **.**
qed

2.10 Definition of Counting Deduction

The theorem Γ $\$\vdash$ φ # Φ = $\sim \varphi$ # Γ $\$\vdash$ $\bot$ # Φ gives rise to another kind of judgement: *how many times can a list of premises* Γ *prove a formula* φ*?*. We call this kind of judgment *counting deduction*. As with measure deduction, bits of Γ get "used up" with each dispatched conclusion.

primrec (**in** *classical-logic*)
counting-deduction :: *'a list* ⇒ *nat* ⇒ *'a* ⇒ *bool* (- #⊢ - - [*60*,*100*,*59*] *60*)
where
Γ #⊢ *0* φ = *True*
| Γ #⊢ (*Suc n*) φ = (∃ Ψ. *mset* (*map snd* Ψ) ⊆# *mset* Γ ∧
map (*uncurry* (⊔)) Ψ :⊢ φ ∧
map (*uncurry* (→)) Ψ @ Γ ⊖ (*map snd* Ψ) #⊢ *n* φ)

2.11 Converting Back and Forth from Counting Deduction to Measure Deduction

We next show how to convert back and forth from counting deduction to measure deduction.

First, we show that trivially counting deduction is a special case of measure deduction.

lemma (**in** *classical-logic*) *counting-deduction-to-measure-deduction*:
Γ #⊢ *n* φ = Γ $\$\vdash$ (*replicate n* φ)
by (*induct n arbitrary*: Γ, *simp*+)

We next prove a few helpful lemmas regarding counting deduction.

lemma (**in** *classical-logic*) *counting-deduction-tautology-weaken*:
assumes ⊢ φ
shows Γ #⊢ *n* φ
proof (*induct n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
hence Γ $\$\vdash$ (φ # *replicate n* φ)

using *assms*
counting-deduction-to-measure-deduction
measure-tautology-right-cancel
by *blast*
hence Γ $\$\vdash$ *replicate* (*Suc n*) φ
by *simp*
then show *?case*
using *counting-deduction-to-measure-deduction*
by *blast*
qed

lemma (**in** *classical-logic*) *counting-deduction-weaken*:
assumes $n \leq m$
and Γ $\#\vdash$ m φ
shows Γ $\#\vdash$ n φ
proof −
have Γ $\$\vdash$ *replicate* m φ
using *assms*(*2*) *counting-deduction-to-measure-deduction*
by *blast*
hence Γ $\$\vdash$ *replicate* n φ
by (*metis append-Nil2*
assms(*1*)
le-iff-add
measure-deduction.simps(*1*)
measure-deduction-generalized-witness
replicate-add)
thus *?thesis*
using *counting-deduction-to-measure-deduction*
by *blast*
qed

lemma (**in** *classical-logic*) *counting-deduction-implication*:
assumes $\vdash$ $\varphi \to \psi$
and Γ $\#\vdash$ n φ
shows Γ $\#\vdash$ n ψ
proof −
have *replicate* n $\psi \preceq$ *replicate* n φ
using *stronger-theory-left-right-cons assms*(*1*)
by (*induct n*, *auto*)
thus *?thesis*
using *assms*(*2*)
measure-stronger-theory-right-antitonic
counting-deduction-to-measure-deduction
by *blast*
qed

Finally, we use Γ $\$\vdash$ φ # Φ = $\sim$ φ # Γ $\$\vdash$ $\bot$ # Φ to prove that measure deduction reduces to counting deduction.

theorem (**in** *classical-logic*) *measure-deduction-to-counting-deduction*:

```
  Γ $⊢ Φ = (∼ Φ @ Γ) #⊢ (length Φ) ⊥
proof −
  have ∀ Ψ. Γ $⊢ (Φ @ Ψ) = (∼ Φ @ Γ) $⊢ (replicate (length Φ) ⊥ @ Ψ)
  proof (induct Φ arbitrary: Γ)
    case Nil
    then show ?case by simp
  next
    case (Cons φ Φ)
    {
      fix Ψ
      have Γ $⊢ ((φ # Φ) @ Ψ) = (∼ φ # Γ) $⊢ (⊥ # Φ @ Ψ)
        using measure-negation-swap by auto
      moreover have mset (Φ @ (⊥ # Ψ)) = mset (⊥ # Φ @ Ψ)
        by simp
      ultimately have Γ $⊢ ((φ # Φ) @ Ψ) = (∼ φ # Γ) $⊢ (Φ @ (⊥ # Ψ))
        by (metis measure-msub-weaken subset-mset.order-refl)
      hence
        Γ $⊢ ((φ # Φ) @ Ψ)
          = (∼ Φ @ (∼ φ # Γ)) $⊢ (replicate (length Φ) ⊥ @ (⊥ # Ψ))
        using Cons
        by blast
      moreover have
        mset (∼ Φ @ (∼ φ # Γ)) = mset (∼ (φ # Φ) @ Γ)
        mset (replicate (length Φ) ⊥ @ (⊥ # Ψ))
          = mset (replicate (length (φ # Φ)) ⊥ @ Ψ)
        by simp+
      ultimately have
       Γ $⊢ ((φ # Φ) @ Ψ) = ∼ (φ # Φ) @ Γ $⊢ (replicate (length (φ # Φ)) ⊥ @
Ψ)
        by (metis
            append.assoc
            append-Cons
            append-Nil
            length-Cons
            replicate-append-same
            list-subtract.simps(1)
            map-ident replicate-Suc
            measure-msub-left-monotonic
            map-list-subtract-mset-containment)
    }
    then show ?case by blast
  qed
  thus ?thesis
    by (metis append-Nil2 counting-deduction-to-measure-deduction)
qed
```

2.12 Measure Deduction Soundess

The last major result for measure deduction we have to show is *soundness*. That is, judgments in measure deduction of lists of formulae can be translated into tautologies for inequalities of finitely additive probability measures over those same formulae (using the same underlying classical logic).

lemma (**in** *classical-logic*) *negated-measure-deduction*:
  $\sim \Gamma$ \$⊢ (φ # Φ) =
    ($\exists$ Ψ. *mset* (*map fst* Ψ) ⊆# *mset* Γ ∧
          $\sim$ (*map* (*uncurry* (\)) Ψ) :⊢ φ ∧
          $\sim$ (*map* (*uncurry* (⊓)) Ψ @ Γ ⊖ (*map fst* Ψ)) \$⊢ Φ)
proof (*rule iffI*)
  assume $\sim \Gamma$ \$⊢ (φ # Φ)
  from *this* **obtain** Ψ **where** Ψ:
    mset (*map snd* Ψ) ⊆# *mset* ($\sim \Gamma$)
    map (*uncurry* (⊔)) Ψ :⊢ φ
    map (*uncurry* (→)) Ψ @ $\sim \Gamma$ ⊖ *map snd* Ψ \$⊢ Φ
    using *measure-deduction.simps*(*2*)
    by *metis*
  from *this* **obtain** Δ **where** Δ:
    mset Δ ⊆# *mset* Γ
    map snd Ψ = $\sim \Delta$
    unfolding *map-negation-def*
    using *mset-sub-map-list-exists* [**where** *f*=$\sim$ **and** Γ=Γ]
    by *metis*
  let *?*Ψ = *zip* Δ (*map fst* Ψ)
  from Δ(*2*) **have** *map fst ?*Ψ = Δ
    unfolding *map-negation-def*
    by (*metis length-map map-fst-zip*)
  with Δ(*1*) **have** *mset* (*map fst ?*Ψ) ⊆# *mset* Γ
    by *simp*
  moreover have ∀ Δ. *map snd* Ψ = $\sim \Delta$ ⟶
                  map (*uncurry* (⊔)) Ψ ⪯ $\sim$ (*map* (*uncurry* (\)) (*zip* Δ (*map fst* Ψ)))
  proof (*induct* Ψ)
    case *Nil*
    then show *?case* **by** *simp*
  next
    case (*Cons* ψ Ψ)
    let *?*ψ = *fst* ψ
    {
      fix Δ
      assume *map snd* (ψ # Ψ) = $\sim \Delta$
      from *this* **obtain** γ **where** γ: $\sim \gamma$ = *snd* ψ γ = *hd* Δ **by** *auto*
      from ‹*map snd* (ψ # Ψ) = $\sim \Delta$› **have** *map snd* Ψ = $\sim$ (*tl* Δ) **by** *auto*
      with *Cons.hyps* **have**
        map (*uncurry* (⊔)) Ψ ⪯ $\sim$ (*map* (*uncurry* (\)) (*zip* (*tl* Δ) (*map fst* Ψ)))
        by *auto*

```
      moreover
      {
        fix ψ γ
        have ⊢ ∼(γ \ ψ) → (ψ ⊔ ∼ γ)
          unfolding disjunction-def
                    subtraction-def
                    conjunction-def
                    negation-def
          by (meson modus-ponens
                    flip-implication
                    hypothetical-syllogism)
      } note tautology = this
      have uncurry (⊔) = (λ ψ. (fst ψ) ⊔ (snd ψ))
        by fastforce
      with γ have uncurry (⊔) ψ = ?ψ ⊔ ∼ γ
        by simp
      with tautology have ⊢ ∼(γ \ ?ψ) → uncurry (⊔) ψ
        by simp
      ultimately have map (uncurry (⊔)) (ψ # Ψ) ⪯
                      ∼ (map (uncurry (\)) ((zip ((hd Δ) # (tl Δ)) (map fst (ψ #
Ψ)))))
        using stronger-theory-left-right-cons γ(2)
        by simp
      hence map (uncurry (⊔)) (ψ # Ψ) ⪯
          ∼ (map (uncurry (\)) (zip Δ (map fst (ψ # Ψ))))
        using ‹map snd (ψ # Ψ) = ∼ Δ› by force
    }
    thus ?case by blast
  qed
  with Ψ(2) Δ(2) have ∼ (map (uncurry (\)) ?Ψ) :⊢ φ
    using stronger-theory-deduction-monotonic by blast
  moreover
  have (map (uncurry (→)) Ψ @ ∼ Γ ⊖ map snd Ψ) ⪯
       ∼ (map (uncurry (⊓)) ?Ψ @ Γ ⊖ (map fst ?Ψ))
  proof −
    from Δ(1) have mset (∼ Γ ⊖ ∼ Δ) = mset (∼ (Γ ⊖ Δ))
      by (simp add: image-mset-Diff)
    hence mset (∼ Γ ⊖ map snd Ψ) = mset (∼ (Γ ⊖ map fst ?Ψ))
      using Ψ(1) Δ(2) ‹map fst ?Ψ = Δ› by simp
    hence (∼ Γ ⊖ map snd Ψ) ⪯ ∼ (Γ ⊖ map fst ?Ψ)
      by (simp add: msub-stronger-theory-intro)
    moreover have ∀ Δ. map snd Ψ = ∼ Δ ⟶
                    map (uncurry (→)) Ψ ⪯ ∼ (map (uncurry (⊓)) (zip Δ (map
fst Ψ)))
    proof (induct Ψ)
      case Nil
      then show ?case by simp
    next
      case (Cons ψ Ψ)
```

```
      let ?ψ = fst ψ
      {
        fix Δ
        assume map snd (ψ # Ψ) = ∼ Δ
        from this obtain γ where γ: ∼ γ = snd ψ γ = hd Δ by auto
        from ‹map snd (ψ # Ψ) = ∼ Δ› have map snd Ψ = ∼ (tl Δ) by auto
        with Cons.hyps have
          map (uncurry (→)) Ψ ≼ ∼ (map (uncurry (⊓)) (zip (tl Δ) (map fst Ψ)))
          by simp
        moreover
        {
          fix ψ γ
          have ⊢ ∼(γ ⊓ ψ) → (ψ → ∼ γ)
            unfolding disjunction-def
                      conjunction-def
                      negation-def
            by (meson modus-ponens
                      flip-implication
                      hypothetical-syllogism)
        } note tautology = this
        have (uncurry (→)) = (λ ψ. (fst ψ) → (snd ψ))
          by fastforce
        with γ have uncurry (→) ψ = ?ψ → ∼ γ
          by simp
        with tautology have ⊢ ∼(γ ⊓ ?ψ) → (uncurry (→)) ψ
          by simp
        ultimately have map (uncurry (→)) (ψ # Ψ) ≼
                        ∼ (map (uncurry (⊓)) ((zip ((hd Δ) # (tl Δ)) (map fst (ψ #
Ψ)))))
          using stronger-theory-left-right-cons γ(2)
          by simp
        hence map (uncurry (→)) (ψ # Ψ) ≼
            ∼ (map (uncurry (⊓)) (zip Δ (map fst (ψ # Ψ))))
          using ‹map snd (ψ # Ψ) = ∼ Δ› by force
      }
      then show ?case by blast
    qed
    ultimately have (map (uncurry (→)) Ψ @ ∼ Γ ⊖ map snd Ψ) ≼
                    (∼ (map (uncurry (⊓)) ?Ψ) @ ∼ (Γ ⊖ (map fst ?Ψ)))
      using stronger-theory-combine Δ(2)
      by metis
    thus ?thesis by simp
  qed
  hence ∼ (map (uncurry (⊓)) ?Ψ @ Γ ⊖ (map fst ?Ψ)) $⊢ Φ
    using Ψ(3) measure-stronger-theory-left-monotonic
    by blast
  ultimately show ∃ Ψ. mset (map fst Ψ) ⊆# mset Γ ∧
                       ∼ (map (uncurry (\)) Ψ) :⊢ φ ∧
                       ∼ (map (uncurry (⊓)) Ψ @ Γ ⊖ (map fst Ψ)) $⊢ Φ
```

```
    by metis
next
  assume ∃Ψ. mset (map fst Ψ) ⊆# mset Γ ∧
              ∼ (map (uncurry (\)) Ψ) :⊢ φ ∧
              ∼ (map (uncurry (⊓)) Ψ @ Γ ⊖ map fst Ψ) $⊢ Φ
  from this obtain Ψ where Ψ:
    mset (map fst Ψ) ⊆# mset Γ
    ∼ (map (uncurry (\)) Ψ) :⊢ φ
    ∼ (map (uncurry (⊓)) Ψ @ Γ ⊖ map fst Ψ) $⊢ Φ
    by auto
  let ?Ψ = zip (map snd Ψ) (∼ (map fst Ψ))
  from Ψ(1) have mset (map snd ?Ψ) ⊆# mset (∼ Γ)
    by (simp, metis image-mset-subseteq-mono multiset.map-comp)
  moreover have ∼ (map (uncurry (\)) Ψ) ≼ map (uncurry (⊔)) ?Ψ
  proof (induct Ψ)
    case Nil
    then show ?case by simp
  next
    case (Cons ψ Ψ)
    let ?γ = fst ψ
    let ?ψ = snd ψ
    {
      fix ψ γ
      have ⊢ (ψ ⊔ ∼ γ) → ∼(γ \ ψ)
        unfolding disjunction-def
                  subtraction-def
                  conjunction-def
                  negation-def
        by (meson modus-ponens
                  flip-implication
                  hypothetical-syllogism)
    } note tautology = this
    have ∼ ∘ uncurry (\) = (λ ψ. ∼ ((fst ψ) \ (snd ψ)))
         uncurry (⊔) = (λ (ψ,γ). ψ ⊔ γ)
      by fastforce+
    with tautology have ⊢ uncurry (⊔) (?ψ, ∼ ?γ) → (∼ ∘ uncurry (\)) ψ
      by fastforce
    with Cons.hyps have
      ((∼ ∘ uncurry (\)) ψ # ∼ (map (uncurry (\)) Ψ)) ≼
       (uncurry (⊔) (?ψ, ∼ ?γ) # map (uncurry (⊔)) (zip (map snd Ψ) (∼ (map
fst Ψ))))
      using stronger-theory-left-right-cons by blast
    thus ?case by simp
  qed
  with Ψ(2) have map (uncurry (⊔)) ?Ψ :⊢ φ
    using stronger-theory-deduction-monotonic by blast
  moreover have ∼ (map (uncurry (⊓)) Ψ @ Γ ⊖ map fst Ψ) ≼
                (map (uncurry (→)) ?Ψ @ ∼ Γ ⊖ map snd ?Ψ)
  proof −
```

```
    have ∼ (map (uncurry (⊓)) Ψ) ⪯ map (uncurry (→)) ?Ψ
    proof (induct Ψ)
      case Nil
      then show ?case by simp
    next
      case (Cons ψ Ψ)
      let ?γ = fst ψ
      let ?ψ = snd ψ
      {
        fix ψ γ
        have ⊢ (ψ → ∼ γ) → ∼(γ ⊓ ψ)
          unfolding disjunction-def
                    conjunction-def
                    negation-def
          by (meson modus-ponens
                    flip-implication
                    hypothetical-syllogism)
      } note tautology = this
      have ∼ ∘ uncurry (⊓) = (λ ψ. ∼ ((fst ψ) ⊓ (snd ψ)))
           uncurry (→) = (λ (ψ,γ). ψ → γ)
        by fastforce+
      with tautology have ⊢ uncurry (→) (?ψ, ∼ ?γ) → (∼ ∘ uncurry (⊓)) ψ
        by fastforce
      with Cons.hyps have
        ((∼ ∘ uncurry (⊓)) ψ # ∼ (map (uncurry (⊓)) Ψ)) ⪯
        (uncurry (→) (?ψ, ∼ ?γ) # map (uncurry (→)) (zip (map snd Ψ) (∼ (map
fst Ψ))))
        using stronger-theory-left-right-cons by blast
      then show ?case by simp
    qed
    moreover have mset (∼ (Γ ⊖ map fst Ψ)) = mset (∼ Γ ⊖ map snd ?Ψ)
      using Ψ(1)
      by (simp add: image-mset-Diff multiset.map-comp)
    hence ∼ (Γ ⊖ map fst Ψ) ⪯ (∼ Γ ⊖ map snd ?Ψ)
      using
        stronger-theory-reflexive
        stronger-theory-right-permutation
      by blast
    ultimately show ?thesis
      using stronger-theory-combine
      by simp
  qed
  hence map (uncurry (→)) ?Ψ @ ∼ Γ ⊖ map snd ?Ψ $⊢ Φ
    using Ψ(3) measure-stronger-theory-left-monotonic by blast
  ultimately show ∼ Γ $⊢ (φ # Φ)
    using measure-deduction.simps(2) by blast
qed

lemma (in probability-logic) measure-deduction-soundness:
```

```
  assumes ∼ Γ $⊢ ∼ Φ
  shows (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←Γ. 𝒫 γ)
proof −
  have ∀ Γ. ∼ Γ $⊢ ∼ Φ ⟶ (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←Γ. 𝒫 γ)
  proof (induct Φ)
    case Nil
    then show ?case
    by (simp, metis (full-types) ex-map-conv probability-non-negative sum-list-nonneg)
  next
    case (Cons φ Φ)
    {
      fix Γ
      assume ∼ Γ $⊢ ∼ (φ # Φ)
      hence ∼ Γ $⊢ (∼ φ # ∼ Φ) by simp
      from this obtain Ψ where Ψ:
        mset (map fst Ψ) ⊆# mset Γ
        ∼ (map (uncurry (\)) Ψ) :⊢ ∼ φ
        ∼ (map (uncurry (⊓)) Ψ @ Γ ⊖ (map fst Ψ)) $⊢ ∼ Φ
        using negated-measure-deduction by blast
      let ?Γ = Γ ⊖ (map fst Ψ)
      let ?Ψ1 = map (uncurry (\)) Ψ
      let ?Ψ2 = map (uncurry (⊓)) Ψ
      have (∑ φ'←Φ. 𝒫 φ') ≤ (∑ φ←(?Ψ2 @ ?Γ). 𝒫 φ)
        using Cons Ψ(3) by blast
      moreover
      have 𝒫 φ ≤ (∑ φ←?Ψ1. 𝒫 φ)
        using Ψ(2)
              biconditional-weaken
              list-deduction-def
              map-negation-list-implication
              set-deduction-base-theory
              implication-list-summation-inequality
        by blast
      ultimately have (∑ φ'←(φ # Φ). 𝒫 φ') ≤ (∑ γ ←(?Ψ1 @ ?Ψ2 @ ?Γ). 𝒫 γ)
        by simp
      moreover have (∑ φ'←(?Ψ1 @ ?Ψ2). 𝒫 φ') = (∑ γ←(map fst Ψ). 𝒫 γ)
      proof (induct Ψ)
        case Nil
        then show ?case by simp
      next
        case (Cons ψ Ψ)
        let ?Ψ1 = map (uncurry (\)) Ψ
        let ?Ψ2 = map (uncurry (⊓)) Ψ
        let ?ψ1 = uncurry (\) ψ
        let ?ψ2 = uncurry (⊓) ψ
        assume (∑ φ'←(?Ψ1 @ ?Ψ2). 𝒫 φ') = (∑ γ←(map fst Ψ). 𝒫 γ)
        moreover
        {
          let ?γ = fst ψ
```

```
        let ?ψ = snd ψ
        have uncurry (\) = (λ ψ. (fst ψ) \ (snd ψ))
             uncurry (⊓) = (λ ψ. (fst ψ) ⊓ (snd ψ))
          by fastforce+
        moreover have 𝒫 ?γ = 𝒫 (?γ \ ?ψ) + 𝒫 (?γ ⊓ ?ψ)
          by (simp add: subtraction-identity)
        ultimately have 𝒫 ?γ = 𝒫 ?ψ₁ + 𝒫 ?ψ₂
          by simp
      }
      moreover have mset (?ψ₁ # ?ψ₂ # (?Ψ₁ @ ?Ψ₂)) =
                    mset (map (uncurry (\)) (ψ # Ψ) @ map (uncurry (⊓)) (ψ #
Ψ))
        (is mset - = mset ?rhs)
        by simp
      hence (∑ φ' ← (?ψ₁ # ?ψ₂ # (?Ψ₁ @ ?Ψ₂)). 𝒫 φ') = (∑ γ ← ?rhs. 𝒫 γ)
        by auto
      ultimately show ?case by simp
    qed
    moreover have mset ((map fst Ψ) @ ?Γ) = mset Γ
      using Ψ(1)
      by simp
    hence (∑ φ'←((map fst Ψ) @ ?Γ). 𝒫 φ') = (∑ γ←Γ. 𝒫 γ)
      by (metis mset-map sum-mset-sum-list)
    ultimately have (∑ φ'←(φ # Φ). 𝒫 φ') ≤ (∑ γ←Γ. 𝒫 γ)
      by simp
  }
  then show ?case by blast
 qed
 thus ?thesis using assms by blast
qed
```

Chapter 3

MaxSAT

We turn now to showing that counting deduction reduces to MaxSAT, the problem of finding the maximal number of satisfiable clauses in a list of clauses.

3.1 Definition of Relative Maximal Clause Collections

Given a list of assumptions Φ and formula φ, we can think of those maximal sublists of Φ that do not prove φ. While in practice we will care about $\varphi = \bot$, we provide a general definition in the more general axiom class *implication-logic*.

definition (**in** *implication-logic*) *relative-maximals* :: *'a list* ⇒ *'a* ⇒ *'a list set* ($\mathcal{M}$)
where
$\mathcal{M}\ \Gamma\ \varphi =$
$\{\ \Phi.\ mset\ \Phi \subseteq\#\ mset\ \Gamma$
$\wedge\ \neg\ \Phi :\vdash \varphi$
$\wedge\ (\forall\ \Psi.\ mset\ \Psi \subseteq\#\ mset\ \Gamma \longrightarrow \neg\ \Psi :\vdash \varphi \longrightarrow length\ \Psi \leq length\ \Phi)\ \}$

lemma (**in** *implication-logic*) *relative-maximals-finite*: *finite* ($\mathcal{M}\ \Gamma\ \varphi$)
proof −
{
fix Φ
assume $\Phi \in \mathcal{M}\ \Gamma\ \varphi$
hence *set* $\Phi \subseteq$ *set* Γ
length $\Phi \leq$ *length* Γ
unfolding *relative-maximals-def*
using *mset-subset-eqD*
length-sub-mset
mset-eq-length
by *fastforce*+
}

```
  hence ℳ Γ φ ⊆ {xs. set xs ⊆ set Γ ∧ length xs ≤ length Γ}
    by auto
  moreover
  have finite {xs. set xs ⊆ set Γ ∧ length xs ≤ length Γ}
    using finite-lists-length-le by blast
  ultimately show ?thesis using rev-finite-subset by auto
qed
```

We know that φ is not a tautology if and only if the set of relative maximal sublists has an element.

```
lemma (in implication-logic) relative-maximals-existence:
  (¬ ⊢ φ) = (∃ Σ. Σ ∈ ℳ Γ φ)
proof (rule iffI)
  assume ¬ ⊢ φ
  show ∃Σ. Σ ∈ ℳ Γ φ
  proof (rule ccontr)
    assume ∄Σ. Σ ∈ ℳ Γ φ
    hence ♢: ∀ Φ. mset Φ ⊆# mset Γ ⟶
                  ¬ Φ :⊢ φ ⟶
                  (∃Ψ. mset Ψ ⊆# mset Γ ∧ ¬ Ψ :⊢ φ ∧ length Ψ > length Φ)
      unfolding relative-maximals-def
      by fastforce
    {
      fix n
      have ∃ Ψ. mset Ψ ⊆# mset Γ ∧ ¬ Ψ :⊢ φ ∧ length Ψ > n
        using ♢
        by (induct n,
            metis
              ‹¬ ⊢ φ›
              list.size(3)
              list-deduction-base-theory
              mset.simps(1)
              subset-mset.zero-le,
            metis
              Nat.lessE
              Suc-less-eq)
    }
    hence ∃ Ψ. mset Ψ ⊆# mset Γ ∧ length Ψ > length Γ
      by auto
    thus False
      using size-mset-mono by fastforce
  qed
next
  assume ∃Σ. Σ ∈ ℳ Γ φ
  thus ¬ ⊢ φ
    unfolding relative-maximals-def
    using list-deduction-weaken
    by blast
qed
```

lemma (**in** *implication-logic*) *relative-maximals-complement-deduction*:
assumes $\Phi \in \mathcal{M}\ \Gamma\ \varphi$
and $\psi \in$ *set* $(\Gamma \ominus \Phi)$
shows $\Phi :\vdash \psi \rightarrow \varphi$
proof (*rule ccontr*)
assume $\neg\ \Phi :\vdash \psi \rightarrow \varphi$
hence $\neg\ (\psi\ \#\ \Phi) :\vdash \varphi$
by (*simp add*: *list-deduction-theorem*)
moreover
have *mset* $\Phi \subseteq\#$ *mset* $\Gamma\ \psi \in\#$ *mset* $(\Gamma \ominus \Phi)$
using *assms*
unfolding *relative-maximals-def*
by (*blast*, *meson in-multiset-in-set*)
hence *mset* $(\psi\ \#\ \Phi) \subseteq\#$ *mset* Γ
by (*simp*, *metis add-mset-add-single*
mset-subset-eq-mono-add-left-cancel
mset-subset-eq-single
subset-mset.add-diff-inverse)
ultimately have *length* $(\psi\ \#\ \Phi) \leq$ *length* (Φ)
using *assms*
unfolding *relative-maximals-def*
by *blast*
thus *False*
by *simp*
qed

lemma (**in** *implication-logic*) *relative-maximals-set-complement* [*simp*]:
assumes $\Phi \in \mathcal{M}\ \Gamma\ \varphi$
shows *set* $(\Gamma \ominus \Phi) =$ *set* $\Gamma -$ *set* Φ
proof (*rule equalityI*)
show *set* $(\Gamma \ominus \Phi) \subseteq$ *set* $\Gamma -$ *set* Φ
proof (*rule subsetI*)
fix ψ
assume $\psi \in$ *set* $(\Gamma \ominus \Phi)$
moreover from *this* **have** $\Phi :\vdash \psi \rightarrow \varphi$
using *assms*
using *relative-maximals-complement-deduction*
by *blast*
hence $\psi \notin$ *set* Φ
using *assms*
list-deduction-modus-ponens
list-deduction-reflection
relative-maximals-def
by *blast*
ultimately show $\psi \in$ *set* $\Gamma -$ *set* Φ
using *list-subtract-set-trivial-upper-bound* [**where** $\Gamma{=}\Gamma$ **and** $\Phi{=}\Phi$]
by *blast*
qed

next
show *set* Γ − *set* Φ ⊆ *set* (Γ ⊖ Φ)
by (*simp add*: *list-subtract-set-difference-lower-bound*)
qed

lemma (**in** *implication-logic*) *relative-maximals-complement-equiv*:
assumes Φ ∈ $\mathcal{M}$ Γ φ
and ψ ∈ *set* Γ
shows Φ :⊢ ψ → φ = (ψ ∉ *set* Φ)
proof (*rule iffI*)
assume Φ :⊢ ψ → φ
thus ψ ∉ *set* Φ
using *assms*(*1*)
list-deduction-modus-ponens
list-deduction-reflection
relative-maximals-def
by *blast*
next
assume ψ ∉ *set* Φ
thus Φ :⊢ ψ → φ
using *assms relative-maximals-complement-deduction*
by *auto*
qed

lemma (**in** *implication-logic*) *maximals-length-equiv*:
assumes Φ ∈ $\mathcal{M}$ Γ φ
and Ψ ∈ $\mathcal{M}$ Γ φ
shows *length* Φ = *length* Ψ
using *assms*
by (*simp add*: *dual-order.antisym relative-maximals-def*)

lemma (**in** *implication-logic*) *maximals-list-subtract-length-equiv*:
assumes Φ ∈ $\mathcal{M}$ Γ φ
and Ψ ∈ $\mathcal{M}$ Γ φ
shows *length* (Γ ⊖ Φ) = *length* (Γ ⊖ Ψ)
proof −
have *length* Φ = *length* Ψ
using *assms maximals-length-equiv*
by *blast*
moreover
have *mset* Φ ⊆# *mset* Γ
mset Ψ ⊆# *mset* Γ
using *assms relative-maximals-def* **by** *blast+*
hence *length* (Γ ⊖ Φ) = *length* Γ − *length* Φ
length (Γ ⊖ Ψ) = *length* Γ − *length* Ψ
by (*metis list-subtract-mset-homomorphism size-Diff-submset size-mset*)+
ultimately show *?thesis* **by** *metis*
qed

We can think of Γ :⊢ φ as saying "the relative maximal sublists of Γ are not

the entire list".

lemma (**in** *implication-logic*) *relative-maximals-max-list-deduction*:
  $\Gamma :\vdash \varphi = (\forall\ \Phi \in \mathcal{M}\ \Gamma\ \varphi.\ 1 \leq \mathit{length}\ (\Gamma \ominus \Phi))$
proof *cases*
  assume $\vdash \varphi$
  hence $\Gamma :\vdash \varphi\ \mathcal{M}\ \Gamma\ \varphi = \{\}$
    unfolding *relative-maximals-def*
    by (*simp add*: *list-deduction-weaken*)+
  then show *?thesis* **by** *blast*
next
  assume $\neg \vdash \varphi$
  from *this* **obtain** Ω **where** Ω: $\Omega \in \mathcal{M}\ \Gamma\ \varphi$
    using *relative-maximals-existence* **by** *blast*
  from *this* **have** *mset* $\Omega \subseteq\#$ *mset* Γ
    unfolding *relative-maximals-def* **by** *blast*
  hence $\diamondsuit$: *length* $(\Gamma \ominus \Omega)$ = *length* Γ − *length* Ω
    by (*metis list-subtract-mset-homomorphism*
          size-Diff-submset
          size-mset)
  show *?thesis*
  proof (*cases* $\Gamma :\vdash \varphi$)
    assume $\Gamma :\vdash \varphi$
    from Ω **have** *mset* $\Omega \subset\#$ *mset* Γ
      by (*metis* (*no-types*, *lifting*)
            Diff-cancel
            Diff-eq-empty-iff
            ‹$\Gamma :\vdash \varphi$›
            list-deduction-monotonic
            relative-maximals-def
            mem-Collect-eq
            mset-eq-setD
            subset-mset.dual-order.not-eq-order-implies-strict)
    hence *length* Ω < *length* Γ
      using *mset-subset-size* **by** *fastforce*
    hence 1 ≤ *length* Γ − *length* Ω
      by (*simp add*: *Suc-leI*)
    with $\diamondsuit$ **have** 1 ≤ *length* $(\Gamma \ominus \Omega)$
      by *simp*
    with ‹$\Gamma :\vdash \varphi$› Ω **show** *?thesis*
      by (*metis maximals-list-subtract-length-equiv*)
  next
    assume $\neg\ \Gamma :\vdash \varphi$
    moreover have *mset* $\Gamma \subseteq\#$ *mset* Γ
      by *simp*
    moreover have *length* $\Omega \leq$ *length* Γ
      using ‹*mset* $\Omega \subseteq\#$ *mset* Γ› *length-sub-mset mset-eq-length*
      by *fastforce*
    ultimately have *length* Ω = *length* Γ
      using Ω

```
    unfolding relative-maximals-def
    by (simp add: dual-order.antisym)
  hence 1 > length (Γ ⊖ Ω)
    using ♢
    by simp
  with ‹¬ Γ :⊢ φ› Ω show ?thesis
    by fastforce
 qed
qed
```

3.2 Definition of MaxSAT

We next turn to defining an abstract form of MaxSAT, which is largest the number of simultaneously satisfiable propositions in a list of propositions.

Unlike conventional MaxSAT, we don't actually work at the *semantic* level, i.e. constructing a model for the Tarski truth relation $\models$. Instead, we just count the elements in a maximal, consistent sublist (i.e., a maximal sub list Σ such that $\neg\ \Sigma :\vdash \bot$) of the list of assumptions Γ we have at hand.

Because we do not work at the semantic level, computing if $\mathit{MaxSAT}\ \Gamma \leq n$ is not in general CoNP-Complete, as it is classically classified [1]. In the special case that the underlying logic is the *classical propositional calculus*, then the complexity is CoNP-Complete. But we could imagine the underlying logic to be linear temporal logic or even first order logic. In such cases the complexity class would be higher in the complexity hierarchy.

```
definition (in implication-logic) relative-MaxSAT :: 'a list ⇒ 'a ⇒ nat (| - |- [45])
 where
  (| Γ |φ) = (if 𝓜 Γ φ = {} then 0 else Max { length Φ | Φ. Φ ∈ 𝓜 Γ φ })

abbreviation (in classical-logic) MaxSAT :: 'a list ⇒ nat
 where
  MaxSAT Γ ≡ | Γ |⊥

definition (in implication-logic) complement-relative-MaxSAT :: 'a list ⇒ 'a ⇒
nat (‖ - ‖- [45])
 where
  (‖ Γ ‖φ) = length Γ − | Γ |φ

lemma (in implication-logic) relative-MaxSAT-intro:
 assumes Φ ∈ 𝓜 Γ φ
 shows length Φ = | Γ |φ
proof −
 have ∀ n ∈ { length Ψ | Ψ. Ψ ∈ 𝓜 Γ φ }. n ≤ length Φ
      length Φ ∈ { length Ψ | Ψ. Ψ ∈ 𝓜 Γ φ }
   using assms relative-maximals-def
   by auto
```

moreover
have *finite* { *length* Ψ | Ψ. Ψ ∈ *M* Γ φ }
using *finite-imageI relative-maximals-finite*
by *simp*
ultimately have *Max* { *length* Ψ | Ψ. Ψ ∈ *M* Γ φ } = *length* Φ
using *Max-eqI*
by *blast*
thus *?thesis*
using *assms relative-MaxSAT-def*
by *auto*
qed

lemma (**in** *implication-logic*) *complement-relative-MaxSAT-intro*:
assumes Φ ∈ *M* Γ φ
shows *length* (Γ ⊖ Φ) = ‖ Γ ‖φ
proof −
have *mset* Φ ⊆# *mset* Γ
using *assms*
unfolding *relative-maximals-def*
by *auto*
moreover from *this* **have** *length* (Γ ⊖ Φ) = *length* Γ − *length* Φ
by (*metis list-subtract-mset-homomorphism size-Diff-submset size-mset*)
ultimately show *?thesis*
unfolding *complement-relative-MaxSAT-def*
by (*metis assms relative-MaxSAT-intro*)
qed

lemma (**in** *implication-logic*) *length-MaxSAT-decomposition*:
length Γ = (| Γ |φ) + ‖ Γ ‖φ
proof (*cases* *M* Γ φ = {})
case *True*
then show *?thesis*
unfolding *relative-MaxSAT-def*
complement-relative-MaxSAT-def
by *simp*
next
case *False*
from *this* **obtain** Φ **where** Φ ∈ *M* Γ φ
by *fast*
moreover from *this* **have** *mset* Φ ⊆# *mset* Γ
unfolding *relative-maximals-def*
by *auto*
moreover from *this* **have** *length* (Γ ⊖ Φ) = *length* Γ − *length* Φ
by (*metis list-subtract-mset-homomorphism size-Diff-submset size-mset*)
ultimately show *?thesis*
unfolding *complement-relative-MaxSAT-def*
using *list-subtract-msub-eq relative-MaxSAT-intro*
by *fastforce*
qed

3.3 Reducing Counting Deduction to MaxSAT

Here we present a major result: counting deduction may be reduced to MaxSAT.

primrec *MaxSAT-optimal-pre-witness* :: *'a list* ⇒ (*'a list* × *'a*) *list* (𝔚)
  where
    𝔚 [] = []
  | 𝔚 (*ψ* # Ψ) = (Ψ, *ψ*) # 𝔚 Ψ

lemma *MaxSAT-optimal-pre-witness-element-inclusion*:
  ∀ (Δ,*δ*) ∈ *set* (𝔚 Ψ). *set* (𝔚 Δ) ⊆ *set* (𝔚 Ψ)
  by (*induct* Ψ, *fastforce*+)

lemma *MaxSAT-optimal-pre-witness-nonelement*:
  assumes *length* Δ ≥ *length* Ψ
  shows (Δ,*δ*) ∉ *set* (𝔚 Ψ)
  using *assms*
proof (*induct* Ψ)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons ψ* Ψ)
  hence Ψ ≠ Δ **by** *auto*
  then show *?case* **using** *Cons* **by** *simp*
qed

lemma *MaxSAT-optimal-pre-witness-distinct*: *distinct* (𝔚 Ψ)
  by (*induct* Ψ, *simp*, *simp add*: *MaxSAT-optimal-pre-witness-nonelement*)

lemma *MaxSAT-optimal-pre-witness-length-iff-eq*:
  ∀ (Δ,*δ*) ∈ *set* (𝔚 Ψ). ∀ (Σ,*σ*) ∈ *set* (𝔚 Ψ). (*length* Δ = *length* Σ) = ((Δ, *δ*) = (Σ,*σ*))
proof (*induct* Ψ)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons ψ* Ψ)
  {
    fix Δ
    fix *δ*
    assume (Δ,*δ*) ∈ *set* (𝔚 (*ψ* # Ψ))
      and *length* Δ = *length* Ψ
    hence (Δ,*δ*) = (Ψ, *ψ*)
      by (*simp add*: *MaxSAT-optimal-pre-witness-nonelement*)
  }
  hence ∀ (Δ,*δ*) ∈ *set* (𝔚 (*ψ* # Ψ)). (*length* Δ = *length* Ψ) = ((Δ,*δ*) = (Ψ,*ψ*))
    by *blast*
  with *Cons* **show** *?case*
    by *auto*

qed

lemma *mset-distinct-msub-down*:
assumes *mset A* $\subseteq\#$ *mset B*
and *distinct B*
shows *distinct A*
using *assms*
by (*meson distinct-append mset-le-perm-append perm-distinct-iff*)

lemma *mset-remdups-set-sub-iff*:
(*mset* (*remdups A*) $\subseteq\#$ *mset* (*remdups B*)) = (*set A* $\subseteq$ *set B*)
proof −
have $\forall$ *B*. (*mset* (*remdups A*) $\subseteq\#$ *mset* (*remdups B*)) = (*set A* $\subseteq$ *set B*)
proof (*induct A*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a A*)
then show *?case*
proof (*cases a* $\in$ *set A*)
case *True*
then show *?thesis* **using** *Cons* **by** *auto*
next
case *False*
{
fix *B*
have (*mset* (*remdups* (*a* # *A*)) $\subseteq\#$ *mset* (*remdups B*)) = (*set* (*a* # *A*) $\subseteq$ *set B*)
proof (*rule iffI*)
assume *assm*: *mset* (*remdups* (*a* # *A*)) $\subseteq\#$ *mset* (*remdups B*)
hence *mset* (*remdups A*) $\subseteq\#$ *mset* (*remdups B*) − {#*a*#}
using *False*
by (*simp add*: *insert-subset-eq-iff*)
hence *mset* (*remdups A*) $\subseteq\#$ *mset* (*remdups* (*removeAll a B*))
by (*metis diff-subset-eq-self*
distinct-remdups
distinct-remove1-removeAll
mset-distinct-msub-down
mset-remove1
set-eq-iff-mset-eq-distinct
set-remdups set-removeAll)
hence *set A* $\subseteq$ *set* (*removeAll a B*)
using *Cons.hyps* **by** *blast*
moreover from *assm False* **have** *a* $\in$ *set B*
using *mset-subset-eq-insertD* **by** *fastforce*
ultimately show *set* (*a* # *A*) $\subseteq$ *set B*
by *auto*
next
assume *assm*: *set* (*a* # *A*) $\subseteq$ *set B*

```
      hence set A ⊆ set (removeAll a B) using False
        by auto
      hence mset (remdups A) ⊆# mset (remdups B) − {#a#}
        by (metis Cons.hyps
                  distinct-remdups
                  mset-remdups-subset-eq
                  mset-remove1 remove-code(1)
                  set-remdups set-remove1-eq
                  set-removeAll
                  subset-mset.dual-order.trans)
      moreover from assm False have a ∈ set B by auto
      ultimately show mset (remdups (a # A)) ⊆# mset (remdups B)
        by (simp add: False insert-subset-eq-iff)
    qed
   }
   then show ?thesis by simp
  qed
 qed
 thus ?thesis by blast
qed

lemma range-characterization:
 (mset X = mset [0..<length X]) = (distinct X ∧ (∀ x ∈ set X. x < length X))
proof (rule iffI)
 assume mset X = mset [0..<length X]
 thus distinct X ∧ (∀ x∈set X. x < length X)
    by (metis atLeastLessThan-iff count-mset-0-iff distinct-count-atmost-1 dis-
tinct-upt set-upt)
next
 assume distinct X ∧ (∀ x∈set X. x < length X)
 moreover
 {
  fix n
  have ∀ X. n = length X ⟶
            distinct X ∧ (∀ x∈set X. x < length X) ⟶
            mset X = mset [0..<length X]
  proof (induct n)
   case 0
   then show ?case by simp
  next
   case (Suc n)
   {
    fix X
    assume A: n + 1 = length X
      and B: distinct X
      and C: ∀ x∈set X. x < length X
    have n ∈ set X
    proof (rule ccontr)
     assume n ∉ set X
```

```
  from A have A': n = length (tl X)
    by simp
  from B have B': distinct (tl X)
    by (simp add: distinct-tl)
  have C': ∀x∈set (tl X). x < length (tl X)
    by (metis
          A
          A'
          C
          ‹n ∉ set X›
          Suc-eq-plus1
          Suc-le-eq
          Suc-le-mono
          le-less
          list.set-sel(2)
          list.size(3)
          nat.simps(3))
  from A' B' C' Suc have mset (tl X) = mset [0..<n]
    by blast
  from A have X = hd X # tl X
    by (metis Suc-eq-plus1 list.exhaust-sel list.size(3) nat.simps(3))
  with B ‹mset (tl X) = mset [0..<n]› have hd X ∉ set [0..<n]
    by (metis distinct.simps(2) mset-eq-setD)
  hence hd X ≥ n by simp
  with C ‹n ∉ set X› ‹X = hd X # tl X› show False
   by (metis A Suc-eq-plus1 Suc-le-eq le-neq-trans list.set-intros(1) not-less)
qed
let ?X' = remove1 n X
have A': n = length ?X'
  by (metis A ‹n ∈ set X› diff-add-inverse2 length-remove1)
have B': distinct ?X'
  by (simp add: B)
have C': ∀x∈set ?X'. x < length ?X'
  by (metis A A' B C
          DiffE
          Suc-eq-plus1
          Suc-le-eq
          Suc-le-mono
          le-neq-trans
          set-remove1-eq
          singletonI)
hence mset ?X' = mset [0..<n]
  using A' B' C' Suc
  by auto
hence mset (n # ?X') = mset [0..<n+1]
  by simp
hence mset X = mset [0..<length X]
  by (metis A ‹n ∈ set X› perm-remove)
}
```

```
    then show ?case by fastforce
  qed
 }
 ultimately show mset X = mset [0..<length X]
  by blast
qed

lemma distinct-pigeon-hole:
 fixes X :: nat list
 assumes distinct X
    and X ≠ []
  shows ∃ n ∈ set X. n + 1 ≥ length X
proof (rule ccontr)
 assume ⋆: ¬ (∃ n ∈ set X. length X ≤ n + 1)
 hence ∀ n ∈ set X. n < length X by fastforce
 hence mset X = mset [0..<length X]
  using assms(1) range-characterization
  by fastforce
 with assms(2) have length X − 1 ∈ set X
  by (metis
       diff-zero
       last-in-set
       last-upt
       length-greater-0-conv
       length-upt mset-eq-setD)
 with ⋆ show False
  by (metis One-nat-def Suc-eq-plus1 Suc-pred le-refl length-pos-if-in-set)
qed

lemma MaxSAT-optimal-pre-witness-pigeon-hole:
 assumes mset Σ ⊆# mset (𝔘 Ψ)
    and Σ ≠ []
  shows ∃ (Δ, δ) ∈ set Σ. length Δ + 1 ≥ length Σ
proof −
 have distinct Σ
  using assms
       MaxSAT-optimal-pre-witness-distinct
       mset-distinct-msub-down
  by blast
 with assms(1) have distinct (map (length ∘ fst) Σ)
 proof (induct Σ)
  case Nil
  then show ?case by simp
 next
  case (Cons σ Σ)
  hence mset Σ ⊆# mset (𝔘 Ψ)
       distinct Σ
   by (metis mset.simps(2) mset-subset-eq-insertD subset-mset-def, simp)
  with Cons.hyps have distinct (map (λa. length (fst a)) Σ) by simp
```

```
    moreover
    obtain δ Δ where σ = (Δ, δ)
      by fastforce
    hence (Δ, δ) ∈ set (𝔘 Ψ)
      using Cons.prems mset-subset-eq-insertD
      by fastforce
    hence ∀ (Σ,σ) ∈ set (𝔘 Ψ). (length Δ = length Σ) = ((Δ, δ) = (Σ, σ))
      using MaxSAT-optimal-pre-witness-length-iff-eq [where Ψ=Ψ]
      by fastforce
    hence ∀ (Σ,σ) ∈ set Σ. (length Δ = length Σ) = ((Δ, δ) = (Σ, σ))
      using ‹mset Σ ⊆# mset (𝔘 Ψ)›
    by (metis (no-types, lifting) Un-iff mset-le-perm-append perm-set-eq set-append)
    hence length (fst σ) ∉ set (map (λa. length (fst a)) Σ)
      using Cons.prems(2) ‹σ = (Δ, δ)›
      by fastforce
    ultimately show ?case by simp
  qed
  moreover have length (map (length ∘ fst) Σ) = length Σ by simp
  moreover have map (length ∘ fst) Σ ≠ [] using assms by simp
  ultimately show ?thesis
    using distinct-pigeon-hole
    by fastforce
qed

abbreviation (in classical-logic)
  MaxSAT-optimal-witness :: 'a ⇒ 'a list ⇒ ('a × 'a) list (𝔚)
  where 𝔚 φ Ξ ≡ map (λ(Ψ,ψ). (Ψ :→ φ, ψ)) (𝔘 Ξ)

abbreviation (in classical-logic)
  disjunction-MaxSAT-optimal-witness :: 'a ⇒ 'a list ⇒ 'a list (𝔚⊔)
  where 𝔚⊔ φ Ψ ≡ map (uncurry (⊔)) (𝔚 φ Ψ)

abbreviation (in classical-logic)
  implication-MaxSAT-optimal-witness :: 'a ⇒ 'a list ⇒ 'a list (𝔚→)
  where 𝔚→ φ Ψ ≡ map (uncurry (→)) (𝔚 φ Ψ)

lemma (in classical-logic) MaxSAT-optimal-witness-conjunction-identity:
  ⊢ ⋀ (𝔚⊔ φ Ψ) ↔ (φ ⊔ ⋀ Ψ)
proof (induct Ψ)
  case Nil
  then show ?case
    unfolding biconditional-def
              disjunction-def
    using axiom-k
          modus-ponens
          verum-tautology
    by (simp, blast)
next
  case (Cons ψ Ψ)
```

have $\vdash (\Psi :\rightarrow \varphi) \leftrightarrow (\bigsqcap \Psi \rightarrow \varphi)$
by (*simp add*: *list-curry-uncurry*)
hence $\vdash \bigsqcap (map\ (uncurry\ (\sqcup))\ (\mathfrak{W}\ \varphi\ (\psi\ \#\ \Psi)))$
$\leftrightarrow ((\bigsqcap \Psi \rightarrow \varphi \sqcup \psi) \sqcap \bigsqcap (map\ (uncurry\ (\sqcup))\ (\mathfrak{W}\ \varphi\ \Psi)))$
unfolding *biconditional-def*
using *conjunction-monotonic*
disjunction-monotonic
by *simp*
moreover have $\vdash ((\bigsqcap \Psi \rightarrow \varphi \sqcup \psi) \sqcap \bigsqcap (map\ (uncurry\ (\sqcup))\ (\mathfrak{W}\ \varphi\ \Psi)))$
$\leftrightarrow ((\bigsqcap \Psi \rightarrow \varphi \sqcup \psi) \sqcap (\varphi \sqcup \bigsqcap \Psi))$
using *Cons.hyps biconditional-conjunction-weaken-rule*
by *blast*
moreover
{
fix $\varphi\ \psi\ \chi$
have $\vdash ((\chi \rightarrow \varphi \sqcup \psi) \sqcap (\varphi \sqcup \chi)) \leftrightarrow (\varphi \sqcup (\psi \sqcap \chi))$
proof −
let $?\varphi = ((\langle\chi\rangle \rightarrow \langle\varphi\rangle \sqcup \langle\psi\rangle) \sqcap (\langle\varphi\rangle \sqcup \langle\chi\rangle)) \leftrightarrow (\langle\varphi\rangle \sqcup (\langle\psi\rangle \sqcap \langle\chi\rangle))$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *fastforce*
hence $\vdash (\!|\ ?\varphi\ |\!)$ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
}
ultimately have $\vdash \bigsqcap (map\ (uncurry\ (\sqcup))\ (\mathfrak{W}\ \varphi\ (\psi\ \#\ \Psi))) \leftrightarrow (\varphi \sqcup (\psi \sqcap \bigsqcap \Psi))$
using *biconditional-transitivity-rule*
by *blast*
then show *?case* **by** *simp*
qed

lemma (**in** *classical-logic*) *MaxSAT-optimal-witness-deduction*:
$\vdash \mathfrak{W}_{\sqcup}\ \varphi\ \Psi :\rightarrow \varphi \leftrightarrow \Psi :\rightarrow \varphi$
proof −
have $\vdash \mathfrak{W}_{\sqcup}\ \varphi\ \Psi :\rightarrow \varphi \leftrightarrow (\bigsqcap (\mathfrak{W}_{\sqcup}\ \varphi\ \Psi) \rightarrow \varphi)$
by (*simp add*: *list-curry-uncurry*)
moreover
{
fix $\alpha\ \beta\ \gamma$
have $\vdash (\alpha \leftrightarrow \beta) \rightarrow ((\alpha \rightarrow \gamma) \leftrightarrow (\beta \rightarrow \gamma))$
proof −
let $?\varphi = (\langle\alpha\rangle \leftrightarrow \langle\beta\rangle) \rightarrow ((\langle\alpha\rangle \rightarrow \langle\gamma\rangle) \leftrightarrow (\langle\beta\rangle \rightarrow \langle\gamma\rangle))$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *fastforce*
hence $\vdash (\!|\ ?\varphi\ |\!)$ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
}
ultimately have $\vdash \mathfrak{W}_{\sqcup}\ \varphi\ \Psi :\rightarrow \varphi \leftrightarrow ((\varphi \sqcup \bigsqcap \Psi) \rightarrow \varphi)$
using *modus-ponens*
biconditional-transitivity-rule

MaxSAT-optimal-witness-conjunction-identity
by *blast*
moreover
{
fix α β
have $\vdash ((\alpha \sqcup \beta) \to \alpha) \leftrightarrow (\beta \to \alpha)$
proof −
let $?\varphi = ((\langle\alpha\rangle \sqcup \langle\beta\rangle) \to \langle\alpha\rangle) \leftrightarrow (\langle\beta\rangle \to \langle\alpha\rangle)$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *fastforce*
hence $\vdash$ ⦇ $?\varphi$ ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
}
ultimately have $\vdash \mathfrak{W}_{\sqcup}\ \varphi\ \Psi :\to \varphi \leftrightarrow (\bigsqcap \Psi \to \varphi)$
using *biconditional-transitivity-rule* **by** *blast*
thus *?thesis*
using *biconditional-symmetry-rule*
biconditional-transitivity-rule
list-curry-uncurry
by *blast*
qed

lemma (**in** *classical-logic*) *optimal-witness-split-identity*:
$\vdash (\mathfrak{W}_{\sqcup}\ \varphi\ (\psi \#\ \Xi)) :\to \varphi \to (\mathfrak{W}_{\to}\ \varphi\ (\psi \#\ \Xi)) :\to \varphi \to \Xi :\to \varphi$
proof (*induct* Ξ)
case *Nil*
have $\vdash ((\varphi \sqcup \psi) \to \varphi) \to ((\varphi \to \psi) \to \varphi) \to \varphi$
proof −
let $?\varphi = ((\langle\varphi\rangle \sqcup \langle\psi\rangle) \to \langle\varphi\rangle) \to ((\langle\varphi\rangle \to \langle\psi\rangle) \to \langle\varphi\rangle) \to \langle\varphi\rangle$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *fastforce*
hence $\vdash$ ⦇ $?\varphi$ ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
then show *?case* **by** *simp*
next
case (*Cons* ξ Ξ)
let $?A = \mathfrak{W}_{\sqcup}\ \varphi\ \Xi :\to \varphi$
let $?B = \mathfrak{W}_{\to}\ \varphi\ \Xi :\to \varphi$
let $?X = \Xi :\to \varphi$
from *Cons.hyps* **have** $\vdash ((?X \sqcup \psi) \to ?A) \to ((?X \to \psi) \to ?B) \to ?X$ **by** *simp*
moreover
have $\vdash (((?X \sqcup \psi) \to ?A) \to ((?X \to \psi) \to ?B) \to ?X)$
$\to ((\xi \to ?X \sqcup \psi) \to (?X \sqcup \xi) \to ?A) \to (((\xi \to ?X) \to \psi) \to (?X \to \xi)$
$\to ?B) \to \xi \to ?X$
proof −
let $?\varphi = ((((\langle ?X\rangle \sqcup \langle\psi\rangle) \to \langle ?A\rangle) \to ((\langle ?X\rangle \to \langle\psi\rangle) \to \langle ?B\rangle) \to \langle ?X\rangle) \to$
$((\langle\xi\rangle \to \langle ?X\rangle \sqcup \langle\psi\rangle) \to (\langle ?X\rangle \sqcup \langle\xi\rangle) \to \langle ?A\rangle) \to$
$((((\langle\xi\rangle \to \langle ?X\rangle) \to \langle\psi\rangle) \to (\langle ?X\rangle \to \langle\xi\rangle) \to \langle ?B\rangle) \to$
$\langle\xi\rangle \to$

$\langle ?X \rangle$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *fastforce*
hence $\vdash$ ⦇ $?\varphi$ ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
ultimately
have $\vdash ((\xi \to ?X \sqcup \psi) \to (?X \sqcup \xi) \to ?A) \to (((\xi \to ?X) \to \psi) \to (?X \to \xi) \to ?B) \to \xi \to ?X$
using *modus-ponens*
by *blast*
thus *?case* **by** *simp*
qed

lemma (**in** *classical-logic*) *disj-conj-impl-duality*:
$\vdash (\varphi \to \chi \sqcap \psi \to \chi) \leftrightarrow ((\varphi \sqcup \psi) \to \chi)$
proof −
let $?\varphi = (\langle\varphi\rangle \to \langle\chi\rangle \sqcap \langle\psi\rangle \to \langle\chi\rangle) \leftrightarrow ((\langle\varphi\rangle \sqcup \langle\psi\rangle) \to \langle\chi\rangle)$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *fastforce*
hence $\vdash$ ⦇ $?\varphi$ ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *weak-disj-of-conj-equiv*:
$(\forall \sigma \in set\ \Sigma.\ \sigma :\vdash \varphi) = \vdash \bigsqcup (map \bigsqcap \Sigma) \to \varphi$
proof (*induct* Σ)
case *Nil*
then show *?case*
by (*simp add*: *ex-falso-quodlibet*)
next
case (*Cons* σ Σ)
have $(\forall \sigma' \in set\ (\sigma \# \Sigma).\ \sigma' :\vdash \varphi) = (\sigma :\vdash \varphi \wedge (\forall \sigma' \in set\ \Sigma.\ \sigma' :\vdash \varphi))$ **by** *simp*
also have ... $= (\vdash \sigma :\to \varphi \wedge \vdash \bigsqcup (map \bigsqcap \Sigma) \to \varphi)$ **using** *Cons.hyps list-deduction-def* **by** *simp*
also have ... $= (\vdash \bigsqcap \sigma \to \varphi \wedge \vdash \bigsqcup (map \bigsqcap \Sigma) \to \varphi)$
using *list-curry-uncurry weak-biconditional-weaken* **by** *blast*
also have ... $= (\vdash \bigsqcap \sigma \to \varphi \sqcap \bigsqcup (map \bigsqcap \Sigma) \to \varphi)$ **by** *simp*
also have ... $= (\vdash (\bigsqcap \sigma \sqcup \bigsqcup (map \bigsqcap \Sigma)) \to \varphi)$
using *disj-conj-impl-duality weak-biconditional-weaken* **by** *blast*
finally show *?case* **by** *simp*
qed

lemma (**in** *classical-logic*) *arbitrary-disj-concat-equiv*:
$\vdash \bigsqcup (\Phi\ @\ \Psi) \leftrightarrow (\bigsqcup \Phi \sqcup \bigsqcup \Psi)$
proof (*induct* Φ)
case *Nil*
then show *?case*
by (*simp*,
meson ex-falso-quodlibet
modus-ponens

biconditional-introduction
disjunction-elimination
disjunction-right-introduction
trivial-implication)
next
case (*Cons* φ Φ)
have $\vdash \bigsqcup\ (\Phi\ @\ \Psi) \leftrightarrow (\bigsqcup\ \Phi \sqcup \bigsqcup\ \Psi) \rightarrow (\varphi \sqcup \bigsqcup\ (\Phi\ @\ \Psi)) \leftrightarrow ((\varphi \sqcup \bigsqcup\ \Phi) \sqcup \bigsqcup\ \Psi)$
proof −
let *?φ* =
$(\langle\bigsqcup\ (\Phi\ @\ \Psi)\rangle \leftrightarrow (\langle\bigsqcup\ \Phi\rangle \sqcup \langle\bigsqcup\ \Psi\rangle)) \rightarrow (\langle\varphi\rangle \sqcup \langle\bigsqcup\ (\Phi\ @\ \Psi)\rangle) \leftrightarrow ((\langle\varphi\rangle \sqcup \langle\bigsqcup\ \Phi\rangle) \sqcup \langle\bigsqcup\ \Psi\rangle)$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop}$ *?φ* **by** *fastforce*
hence $\vdash$ ⦇ *?φ* ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
then show *?case* **using** *Cons modus-ponens* **by** *simp*
qed

lemma (**in** *classical-logic*) *arbitrary-conj-concat-equiv*:
$\vdash \bigsqcap\ (\Phi\ @\ \Psi) \leftrightarrow (\bigsqcap\ \Phi \sqcap \bigsqcap\ \Psi)$
proof (*induct* Φ)
case *Nil*
then show *?case*
by (*simp*,
meson modus-ponens
biconditional-introduction
conjunction-introduction
conjunction-right-elimination
verum-tautology)
next
case (*Cons* φ Φ)
have $\vdash \bigsqcap\ (\Phi\ @\ \Psi) \leftrightarrow (\bigsqcap\ \Phi \sqcap \bigsqcap\ \Psi) \rightarrow (\varphi \sqcap \bigsqcap\ (\Phi\ @\ \Psi)) \leftrightarrow ((\varphi \sqcap \bigsqcap\ \Phi) \sqcap \bigsqcap\ \Psi)$
proof −
let *?φ* =
$(\langle\bigsqcap\ (\Phi\ @\ \Psi)\rangle \leftrightarrow (\langle\bigsqcap\ \Phi\rangle \sqcap \langle\bigsqcap\ \Psi\rangle)) \rightarrow (\langle\varphi\rangle \sqcap \langle\bigsqcap\ (\Phi\ @\ \Psi)\rangle) \leftrightarrow ((\langle\varphi\rangle \sqcap \langle\bigsqcap\ \Phi\rangle) \sqcap \langle\bigsqcap\ \Psi\rangle)$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop}$ *?φ* **by** *fastforce*
hence $\vdash$ ⦇ *?φ* ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
then show *?case* **using** *Cons modus-ponens* **by** *simp*
qed

lemma (**in** *classical-logic*) *conj-absorption*:
assumes $\chi \in$ *set* Φ
shows $\vdash \bigsqcap\ \Phi \leftrightarrow (\chi \sqcap \bigsqcap\ \Phi)$
using *assms*

proof (*induct* Φ)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons* φ Φ)
  then show *?case*
  proof (*cases* $\varphi = \chi$)
    case *True*
    then show *?thesis*
      by (*simp*,
          metis biconditional-def
                implication-distribution
                trivial-implication
                weak-biconditional-weaken
                weak-conjunction-deduction-equivalence)
  next
    case *False*
    then show *?thesis*
      by (*metis Cons.prems*
                arbitrary-conjunction.simps(*2*)
                modus-ponens
                arbitrary-conjunction-antitone
                biconditional-introduction
                remdups.simps(*2*)
                set-remdups
                set-subset-Cons)
  qed
qed

lemma (**in** *classical-logic*) *conj-extract*: $\vdash \bigsqcup\ (map\ ((\sqcap)\ \varphi)\ \Psi) \leftrightarrow (\varphi \sqcap \bigsqcup\ \Psi)$
proof (*induct* Ψ)
  case *Nil*
  then show *?case*
    by (*simp add*: *ex-falso-quodlibet biconditional-def conjunction-right-elimination*)
next
  case (*Cons* ψ Ψ)
  have $\vdash \bigsqcup\ (map\ ((\sqcap)\ \varphi)\ \Psi) \leftrightarrow (\varphi \sqcap \bigsqcup\ \Psi)$
        $\rightarrow ((\varphi \sqcap \psi) \sqcup \bigsqcup\ (map\ ((\sqcap)\ \varphi)\ \Psi)) \leftrightarrow (\varphi \sqcap (\psi \sqcup \bigsqcup\ \Psi))$
  proof −
    let *?*$\varphi = \langle \bigsqcup\ (map\ ((\sqcap)\ \varphi)\ \Psi)\rangle \leftrightarrow (\langle\varphi\rangle \sqcap \langle \bigsqcup\ \Psi\rangle)$
            $\rightarrow ((\langle\varphi\rangle \sqcap \langle\psi\rangle) \sqcup \langle \bigsqcup\ (map\ ((\sqcap)\ \varphi)\ \Psi)\rangle) \leftrightarrow (\langle\varphi\rangle \sqcap (\langle\psi\rangle \sqcup \langle \bigsqcup\ \Psi\rangle))$
    have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop}$ *?*φ **by** *fastforce*
    hence $\vdash$ ⦇ *?*φ ⦈ **using** *propositional-semantics* **by** *blast*
    thus *?thesis* **by** *simp*
  qed
  then show *?case* **using** *Cons modus-ponens* **by** *simp*
qed

lemma (**in** *classical-logic*) *conj-multi-extract*:

$\vdash \bigsqcup\ (map\ \bigsqcap\ (map\ ((@)\ \Delta)\ \Sigma)) \leftrightarrow (\bigsqcap\ \Delta \sqcap \bigsqcup\ (map\ \bigsqcap\ \Sigma))$
proof (*induct* Σ)
case *Nil*
then show *?case*
by (*simp*, *metis list.simps(8) arbitrary-disjunction.simps(1) conj-extract*)
next
case (*Cons* σ Σ)
moreover have
$\vdash \bigsqcup\ (map\ \bigsqcap\ (map\ ((@)\ \Delta)\ \Sigma)) \leftrightarrow (\bigsqcap\ \Delta \sqcap \bigsqcup\ (map\ \bigsqcap\ \Sigma))$
$\rightarrow \bigsqcap\ (\Delta\ @\ \sigma) \leftrightarrow (\bigsqcap\ \Delta \sqcap \bigsqcap\ \sigma)$
$\rightarrow (\bigsqcap\ (\Delta\ @\ \sigma) \sqcup \bigsqcup\ (map\ (\bigsqcap \circ (@)\ \Delta)\ \Sigma)) \leftrightarrow (\bigsqcap\ \Delta \sqcap (\bigsqcap\ \sigma \sqcup \bigsqcup\ (map\ \bigsqcap\ \Sigma)))$
proof −
let *?φ* =
$\langle \bigsqcup\ (map\ \bigsqcap\ (map\ ((@)\ \Delta)\ \Sigma))\rangle \leftrightarrow (\langle \bigsqcap\ \Delta\rangle \sqcap \langle \bigsqcup\ (map\ \bigsqcap\ \Sigma)\rangle)$
$\rightarrow \langle \bigsqcap\ (\Delta\ @\ \sigma)\rangle \leftrightarrow (\langle \bigsqcap\ \Delta\rangle \sqcap \langle \bigsqcap\ \sigma\rangle)$
$\rightarrow (\langle \bigsqcap\ (\Delta\ @\ \sigma)\rangle \sqcup \langle \bigsqcup\ (map\ (\bigsqcap \circ (@)\ \Delta)\ \Sigma)\rangle) \leftrightarrow (\langle \bigsqcap\ \Delta\rangle \sqcap (\langle \bigsqcap\ \sigma\rangle \sqcup \langle \bigsqcup\ (map\ \bigsqcap\ \Sigma)\rangle))$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *fastforce*
hence $\vdash$ ⦇ *?φ* ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
hence
$\vdash (\bigsqcap\ (\Delta\ @\ \sigma) \sqcup \bigsqcup\ (map\ (\bigsqcap \circ (@)\ \Delta)\ \Sigma)) \leftrightarrow (\bigsqcap\ \Delta \sqcap (\bigsqcap\ \sigma \sqcup \bigsqcup\ (map\ \bigsqcap\ \Sigma)))$
using *Cons.hyps arbitrary-conj-concat-equiv modus-ponens* **by** *blast*
then show *?case* **by** *simp*
qed

lemma (**in** *classical-logic*) *extract-inner-concat*:
$\vdash \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ (@)\ \Delta))\ \Psi) \leftrightarrow (\bigsqcap\ (map\ snd\ \Delta) \sqcap \bigsqcup\ (map\ (\bigsqcap \circ map\ snd)\ \Psi))$
proof (*induct* Δ)
case *Nil*
then show *?case*
by (*simp*,
meson modus-ponens
biconditional-introduction
conjunction-introduction
conjunction-right-elimination
verum-tautology)
next
case (*Cons* χ Δ)
let *?*$\Delta' = map\ snd\ \Delta$
let *?*$\chi' = snd\ \chi$
let *?*$\Pi = \lambda\varphi.\ \bigsqcap\ (map\ snd\ \varphi)$
let *?*$\Pi\Delta = \lambda\varphi.\ \bigsqcap\ (?\Delta'\ @\ map\ snd\ \varphi)$
from *Cons* **have**
$\vdash \bigsqcup\ (map\ ?\Pi\Delta\ \Psi) \leftrightarrow (\bigsqcap\ ?\Delta' \sqcap \bigsqcup\ (map\ ?\Pi\ \Psi))$

```
    by auto
  moreover have ⋆: map (λφ. ?χ' ⊓ ?ΠΔ φ) = map ((⊓) ?χ') ∘ map ?ΠΔ
    by fastforce
  have ⨆ (map (λφ. ?χ' ⊓ ?ΠΔ φ) Ψ) = ⨆ (map ((⊓) ?χ') (map ?ΠΔ Ψ))
    by (simp add: ⋆)
  hence
    ⊢ ⨆ (map (λφ. ?χ' ⊓ ?ΠΔ φ) Ψ) ↔ (?χ' ⊓ ⨆ (map (λφ. ?ΠΔ φ) Ψ))
    using conj-extract by presburger
  moreover have
    ⊢ ⨆ (map ?ΠΔ Ψ) ↔ (⨅ ?Δ' ⊓ ⨆ (map ?Π Ψ))
    → ⨆ (map (λφ. ?χ' ⊓ ?ΠΔ φ) Ψ) ↔ (?χ' ⊓ ⨆ (map ?ΠΔ Ψ))
    → ⨆ (map (λφ. ?χ' ⊓ ?ΠΔ φ) Ψ) ↔ ((?χ' ⊓ ⨅ ?Δ') ⊓ ⨆ (map ?Π Ψ))
  proof −
    let ?φ = ⟨⨆ (map ?ΠΔ Ψ)⟩ ↔ (⟨⨅ ?Δ'⟩ ⊓ ⟨⨆ (map ?Π Ψ)⟩)
          → ⟨⨆ (map (λφ. ?χ' ⊓ ?ΠΔ φ) Ψ)⟩ ↔ (⟨?χ'⟩ ⊓ ⟨⨆ (map ?ΠΔ Ψ)⟩)
           → ⟨⨆ (map (λφ. ?χ' ⊓ ?ΠΔ φ) Ψ)⟩ ↔ ((⟨?χ'⟩ ⊓ ⟨⨅ ?Δ'⟩) ⊓ ⟨⨆
(map ?Π Ψ)⟩)
    have ∀ 𝔐. 𝔐 ⊨_prop ?φ by fastforce
    hence ⊢ ⦇ ?φ ⦈ using propositional-semantics by blast
    thus ?thesis by simp
  qed
  ultimately have ⊢ ⨆ (map (λφ. ?χ' ⊓ ⨅ (?Δ' @ map snd φ)) Ψ)
              ↔ ((?χ' ⊓ ⨅ ?Δ') ⊓ ⨆ (map (λφ. ⨅ (map snd φ)) Ψ))
    using modus-ponens by blast
  thus ?case by simp
qed

lemma (in classical-logic) extract-inner-concat-remdups:
  ⊢ ⨆ (map (⨅ ∘ (map snd ∘ remdups ∘ (@) Δ)) Ψ) ↔
    (⨅ (map snd Δ) ⊓ ⨆ (map (⨅ ∘ (map snd ∘ remdups)) Ψ))
proof −
  have ∀ Ψ. ⊢ ⨆ (map (⨅ ∘ (map snd ∘ remdups ∘ (@) Δ)) Ψ) ↔
             (⨅ (map snd Δ) ⊓ ⨆ (map (⨅ ∘ (map snd ∘ remdups)) Ψ))
  proof (induct Δ)
    case Nil
    then show ?case
      by (simp,
          meson modus-ponens
                biconditional-introduction
                conjunction-introduction
                conjunction-right-elimination
                verum-tautology)
  next
    case (Cons δ Δ)
    {
      fix Ψ
      have ⊢   ⨆ (map (⨅ ∘ (map snd ∘ remdups ∘ (@) (δ # Δ))) Ψ)
           ↔ (⨅ (map snd (δ # Δ)) ⊓ ⨆ (map (⨅ ∘ (map snd ∘ remdups)) Ψ))
      proof (cases δ ∈ set Δ)
```

assume $\delta \in set\ \Delta$
have
$\vdash \quad \bigsqcap (map\ snd\ \Delta) \leftrightarrow (snd\ \delta \sqcap \bigsqcap (map\ snd\ \Delta))$
$\rightarrow \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ \Delta))\ \Psi)$
$\leftrightarrow (\bigsqcap (map\ snd\ \Delta) \sqcap \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi))$
$\rightarrow \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ \Delta))\ \Psi)$
$\leftrightarrow ((snd\ \delta \sqcap \bigsqcap (map\ snd\ \Delta)) \sqcap \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi))$
proof −
let $?\varphi = \quad \langle \bigsqcap (map\ snd\ \Delta) \rangle \leftrightarrow (\langle snd\ \delta \rangle \sqcap \langle \bigsqcap (map\ snd\ \Delta) \rangle)$
$\rightarrow \langle \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ \Delta))\ \Psi) \rangle$
$\leftrightarrow (\langle \bigsqcap (map\ snd\ \Delta) \rangle \sqcap \langle \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi) \rangle)$
$\rightarrow \langle \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ \Delta))\ \Psi) \rangle$
$\leftrightarrow ((\langle snd\ \delta \rangle \sqcap \langle \bigsqcap (map\ snd\ \Delta) \rangle) \sqcap \langle \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi) \rangle)$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *fastforce*
hence $\vdash (\!|\ ?\varphi\ |\!)$ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
moreover have $\vdash \bigsqcap (map\ snd\ \Delta) \leftrightarrow (snd\ \delta \sqcap \bigsqcap (map\ snd\ \Delta))$
by (*simp add*: ‹$\delta \in set\ \Delta$› *conj-absorption*)
ultimately have
$\vdash \quad \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ \Delta))\ \Psi)$
$\leftrightarrow ((snd\ \delta \sqcap \bigsqcap (map\ snd\ \Delta)) \sqcap \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi))$
using *Cons.hyps modus-ponens* **by** *blast*
moreover have $map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta) = map\ snd \circ remdups \circ (@)\ \Delta$
using ‹$\delta \in set\ \Delta$› **by** *fastforce*
ultimately show *?thesis* **using** *Cons* **by** *simp*
next
assume $\delta \notin set\ \Delta$
hence †:
$\bigsqcap \circ (map\ snd \circ remdups) = (\lambda\psi.\ \bigsqcap (map\ snd\ (remdups\ \psi)))$
$(\lambda\psi.\ \bigsqcap (map\ snd\ (if\ \delta \in set\ \psi\ then\ remdups\ (\Delta\ @\ \psi)\ else\ \delta\ \#\ remdups\ (\Delta\ @\ \psi))))$
$= \bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta))$
by *fastforce*+
show *?thesis*
proof (*induct* Ψ)
case *Nil*
then show *?case*
by (*simp*, *metis list.simps*(*8*) *arbitrary-disjunction.simps*(*1*) *conj-extract*)
next
case (*Cons* ψ Ψ)
have $\vdash \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ \Delta))\ [\psi])$
$\leftrightarrow (\bigsqcap (map\ snd\ \Delta) \sqcap \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ [\psi]))$
using ‹$\forall \Psi.\ \vdash \quad \bigsqcup (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ \Delta))\ \Psi)$

$\leftrightarrow (\bigsqcap\ (map\ snd\ \Delta) \sqcap \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi))\rangle$
by *blast*
hence
$\vdash\ (\bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi))) \sqcup \bot)$
$\leftrightarrow (\bigsqcap\ (map\ snd\ \Delta) \sqcap \bigsqcap\ (map\ snd\ (remdups\ \psi)) \sqcup \bot)$
by *simp*
hence $\star$:
$\vdash \bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi))) \leftrightarrow (\bigsqcap\ (map\ snd\ \Delta) \sqcap \bigsqcap\ (map\ snd\ (remdups\ \psi)))$
by (*metis*
(*no-types*, *opaque-lifting*)
biconditional-conjunction-weaken-rule
biconditional-symmetry-rule
biconditional-transitivity-rule
disjunction-def
double-negation-biconditional
negation-def)
have $\vdash\ \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi)$
$\leftrightarrow (\bigsqcap\ (map\ snd\ (\delta\ \#\ \Delta)) \sqcap \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi))$
using *Cons* **by** *blast*
hence $\diamondsuit$: $\vdash\ \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi)$
$\leftrightarrow ((snd\ \delta \sqcap \bigsqcap\ (map\ snd\ \Delta)) \sqcap \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi))$
by *simp*
show *?case*
proof (*cases* $\delta \in set\ \psi$)
assume $\delta \in set\ \psi$
have $snd\ \delta \in set\ (map\ snd\ (remdups\ \psi))$
using $\langle \delta \in set\ \psi \rangle$ **by** *auto*
hence $\spadesuit$: $\vdash \bigsqcap\ (map\ snd\ (remdups\ \psi)) \leftrightarrow (snd\ \delta \sqcap \bigsqcap\ (map\ snd\ (remdups\ \psi)))$
using *conj-absorption* **by** *blast*
have
$\vdash\ (\bigsqcap\ (map\ snd\ (remdups\ \psi)) \leftrightarrow (snd\ \delta \sqcap \bigsqcap\ (map\ snd\ (remdups\ \psi))))$
$\rightarrow (\bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi)$
$\leftrightarrow ((snd\ \delta \sqcap \bigsqcap\ (map\ snd\ \Delta)) \sqcap \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi)))$
$\rightarrow (\bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi))) \leftrightarrow (\bigsqcap\ (map\ snd\ \Delta) \sqcap \bigsqcap\ (map\ snd\ (remdups\ \psi))))$
$\rightarrow\ (\bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi)))$
$\sqcup \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi))$
$\leftrightarrow ((snd\ \delta \sqcap \bigsqcap\ (map\ snd\ \Delta))$
$\sqcap (\bigsqcap\ (map\ snd\ (remdups\ \psi)) \sqcup \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi)))$
proof $-$
let $?\varphi =$
$(\langle \bigsqcap\ (map\ snd\ (remdups\ \psi)) \rangle \leftrightarrow (\langle snd\ \delta \rangle \sqcap \langle \bigsqcap\ (map\ snd\ (remdups$

$\psi))\rangle\rangle))$
$\rightarrow \quad (\langle \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi)\rangle$
$\leftrightarrow ((\langle snd\ \delta\rangle \sqcap \langle \bigsqcap\ (map\ snd\ \Delta)\rangle) \sqcap \langle \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi)\rangle))$
$\rightarrow \quad (\langle \bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi)))\rangle$
$\leftrightarrow (\langle \bigsqcap\ (map\ snd\ \Delta)\rangle \sqcap \langle \bigsqcap\ (map\ snd\ (remdups\ \psi))\rangle))$
$\rightarrow \quad (\langle \bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi)))\rangle$
$\sqcup \langle \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi)\rangle)$
$\leftrightarrow ((\langle snd\ \delta\rangle \sqcap \langle \bigsqcap\ (map\ snd\ \Delta)\rangle)$
$\sqcap (\langle \bigsqcap\ (map\ snd\ (remdups\ \psi))\rangle \sqcup \langle \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi)\rangle))$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *fastforce*
hence $\vdash$ (| $?\varphi$ |) **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
hence
$\vdash \quad (\bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi)))$
$\sqcup \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi))$
$\leftrightarrow ((snd\ \delta \sqcap \bigsqcap\ (map\ snd\ \Delta))$
$\sqcap (\bigsqcap\ (map\ snd\ (remdups\ \psi)) \sqcup \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi)))$
using $\star\ \diamondsuit\ \spadesuit$ *modus-ponens* **by** *blast*
thus *?thesis* **using** ‹$\delta \notin set\ \Delta$› ‹$\delta \in set\ \psi$›
by (*simp add*: †)
next
assume $\delta \notin set\ \psi$
have
$\vdash \quad (\bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi)$
$\leftrightarrow ((snd\ \delta \sqcap \bigsqcap\ (map\ snd\ \Delta)) \sqcap \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi)))$
$\rightarrow (\bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi))) \leftrightarrow (\bigsqcap\ (map\ snd\ \Delta) \sqcap \bigsqcap\ (map\ snd\ (remdups\ \psi))))$
$\rightarrow \quad ((snd\ \delta \sqcap \bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi))))$
$\sqcup \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi))$
$\leftrightarrow ((snd\ \delta \sqcap \bigsqcap\ (map\ snd\ \Delta))$
$\sqcap (\bigsqcap\ (map\ snd\ (remdups\ \psi)) \sqcup \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi)))$
proof −
let $?\varphi =$
$(\langle \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi)\rangle$
$\leftrightarrow ((\langle snd\ \delta\rangle \sqcap \langle \bigsqcap\ (map\ snd\ \Delta)\rangle) \sqcap \langle \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi)\rangle))$
$\rightarrow \quad (\langle \bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi)))\rangle$
$\leftrightarrow (\langle \bigsqcap\ (map\ snd\ \Delta)\rangle \sqcap \langle \bigsqcap\ (map\ snd\ (remdups\ \psi))\rangle))$
$\rightarrow \quad ((\langle snd\ \delta\rangle \sqcap \langle \bigsqcap\ (map\ snd\ (remdups\ (\Delta\ @\ \psi)))\rangle)$
$\sqcup \langle \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups \circ (@)\ (\delta\ \#\ \Delta)))\ \Psi)\rangle)$
$\leftrightarrow ((\langle snd\ \delta\rangle \sqcap \langle \bigsqcap\ (map\ snd\ \Delta)\rangle)$
$\sqcap (\langle \bigsqcap\ (map\ snd\ (remdups\ \psi))\rangle \sqcup \langle \bigsqcup\ (map\ (\bigsqcap \circ (map\ snd \circ remdups))\ \Psi)\rangle))$

```
      have ∀ 𝔐. 𝔐 ⊨_prop ?φ by fastforce
      hence ⊢ ⦇ ?φ ⦈ using propositional-semantics by blast
      thus ?thesis by simp
    qed
    hence
      ⊢   ((snd δ ⊓ ⨅ (map snd (remdups (Δ @ ψ))))
            ⊔ ⨆ (map (⨅ ∘ (map snd ∘ remdups ∘ (@) (δ # Δ))) Ψ))
        ↔ ((snd δ ⊓ ⨅ (map snd Δ))
            ⊓ (⨅ (map snd (remdups ψ)) ⊔ ⨆ (map (⨅ ∘ (map snd ∘ remdups)) Ψ)))
      using ⋆ ♢ modus-ponens by blast
    then show ?thesis using ‹δ ∉ set ψ› ‹δ ∉ set Δ› by (simp add: †)
   qed
  qed
 qed
}
then show ?case by fastforce
qed
thus ?thesis by blast
qed

lemma (in classical-logic) optimal-witness-list-intersect-biconditional:
 assumes mset Ξ ⊆# mset Γ
   and mset Φ ⊆# mset (Γ ⊖ Ξ)
   and mset Ψ ⊆# mset (𝔚→ φ Ξ)
 shows ∃ Σ. ⊢ ((Φ @ Ψ) :→ φ) ↔ (⨆ (map ⨅ Σ) → φ)
         ∧ (∀ σ ∈ set Σ. mset σ ⊆# mset Γ ∧ length σ + 1 ≥ length (Φ @ Ψ))
proof −
 have ∃ Σ. ⊢ (Ψ :→ φ) ↔ (⨆ (map ⨅ Σ) → φ)
         ∧ (∀ σ ∈ set Σ. mset σ ⊆# mset Ξ ∧ length σ + 1 ≥ length Ψ)
 proof −
   from assms(3) obtain Ψ₀ :: ('a list × 'a) list where Ψ₀:
     mset Ψ₀ ⊆# mset (𝔚 Ξ)
     map (λ(Ψ,ψ). (Ψ :→ φ → ψ)) Ψ₀ = Ψ
     using mset-sub-map-list-exists by fastforce
   let ?Π_C = λ (Δ,δ) Σ. (map ((#) (Δ, δ)) Σ) @ (map ((@) (𝔚 Δ)) Σ)
   let ?T_Σ = λ Ψ. foldr ?Π_C Ψ [[]]
   let ?Σ = map (map snd ∘ remdups) (?T_Σ Ψ₀)
   have I: ⊢ (Ψ :→ φ) ↔ (⨆ (map ⨅ ?Σ) → φ)
   proof −
     let ?Σ_α = map (map snd) (?T_Σ Ψ₀)
     let ?Ψ' = map (λ(Ψ,ψ). (Ψ :→ φ → ψ)) Ψ₀
     {
       fix Ψ :: ('a list × 'a) list
       let ?Σ_α = map (map snd) (?T_Σ Ψ)
       let ?Σ = map (map snd ∘ remdups) (?T_Σ Ψ)
       have ⊢ (⨆ (map ⨅ ?Σ_α) → φ) ↔ (⨆ (map ⨅ ?Σ) → φ)
       proof (induct Ψ)
         case Nil
```

then show *?case* **by** (*simp add*: *biconditional-reflection*)
next
case (*Cons* $\Delta\delta$ Ψ)
let *?*Δ = *fst* $\Delta\delta$
let *?*δ = *snd* $\Delta\delta$
let *?*Σ_α = *map* (*map snd*) (*?*T_Σ Ψ)
let *?*Σ = *map* (*map snd* ∘ *remdups*) (*?*T_Σ Ψ)
let *?*$\Sigma_\alpha{}'$ = *map* (*map snd*) (*?*T_Σ ((*?*Δ,*?*δ) # Ψ))
let *?*Σ' = *map* (*map snd* ∘ *remdups*) (*?*T_Σ ((*?*Δ,*?*δ) # Ψ))
{
fix Δ :: *'a list*
fix δ :: *'a*
let *?*$\Sigma_\alpha{}'$ = *map* (*map snd*) (*?*T_Σ ((Δ,δ) # Ψ))
let *?*Σ' = *map* (*map snd* ∘ *remdups*) (*?*T_Σ ((Δ,δ) # Ψ))
let *?*Φ = *map* (*map snd* ∘ (@) [(Δ, δ)]) (*?*T_Σ Ψ)
let *?*Ψ = *map* (*map snd* ∘ (@) ($\mathfrak{V}$ Δ)) (*?*T_Σ Ψ)
let *?*Δ = *map* (*map snd* ∘ *remdups* ∘ (@) [(Δ, δ)]) (*?*T_Σ Ψ)
let *?*Ω = *map* (*map snd* ∘ *remdups* ∘ (@) ($\mathfrak{V}$ Δ)) (*?*T_Σ Ψ)
have ⊢ ($\bigsqcup$ (*map* $\bigsqcap$ *?*Φ @ *map* $\bigsqcap$ *?*Ψ) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?*Φ) ⊔ $\bigsqcup$ (*map* $\bigsqcap$ *?*Ψ))) →
($\bigsqcup$ (*map* $\bigsqcap$ *?*Δ @ *map* $\bigsqcap$ *?*Ω) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?*Δ) ⊔ $\bigsqcup$ (*map* $\bigsqcap$ *?*Ω))) →
($\bigsqcup$ (*map* $\bigsqcap$ *?*Φ) ↔ ($\bigsqcap$ [δ] ⊓ $\bigsqcup$ (*map* $\bigsqcap$ *?*Σ_α))) →
($\bigsqcup$ (*map* $\bigsqcap$ *?*Ψ) ↔ ($\bigsqcap$ Δ ⊓ $\bigsqcup$ (*map* $\bigsqcap$ *?*Σ_α))) →
($\bigsqcup$ (*map* $\bigsqcap$ *?*Δ) ↔ ($\bigsqcap$ [δ] ⊓ $\bigsqcup$ (*map* $\bigsqcap$ *?*Σ))) →
($\bigsqcup$ (*map* $\bigsqcap$ *?*Ω) ↔ ($\bigsqcap$ Δ ⊓ $\bigsqcup$ (*map* $\bigsqcap$ *?*Σ))) →
(($\bigsqcup$ (*map* $\bigsqcap$ *?*Σ_α) → φ) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?*Σ) → φ)) →
(($\bigsqcup$ (*map* $\bigsqcap$ *?*Φ @ *map* $\bigsqcap$ *?*Ψ) → φ) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?*Δ @ *map* $\bigsqcap$ *?*Ω) → φ))
proof −
let *?*φ =
(⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Φ @ *map* $\bigsqcap$ *?*Ψ)⟩ ↔ (⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Φ)⟩ ⊔ ⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Ψ)⟩)) →
(⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Δ @ *map* $\bigsqcap$ *?*Ω)⟩ ↔ (⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Δ)⟩ ⊔ ⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Ω)⟩)) →
(⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Φ)⟩ ↔ (⟨$\bigsqcap$ [δ]⟩ ⊓ ⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Σ_α)⟩)) →
(⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Ψ)⟩ ↔ (⟨$\bigsqcap$ Δ⟩ ⊓ ⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Σ_α)⟩)) →
(⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Δ)⟩ ↔ (⟨$\bigsqcap$ [δ]⟩ ⊓ ⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Σ)⟩)) →
(⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Ω)⟩ ↔ (⟨$\bigsqcap$ Δ⟩ ⊓ ⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Σ)⟩)) →
((⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Σ_α)⟩ → ⟨φ⟩) ↔ (⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Σ)⟩ → ⟨φ⟩)) →
((⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Φ @ *map* $\bigsqcap$ *?*Ψ)⟩ → ⟨φ⟩) ↔ (⟨$\bigsqcup$ (*map* $\bigsqcap$ *?*Δ @ *map* $\bigsqcap$ *?*Ω)⟩ → ⟨φ⟩))
have ∀$\mathfrak{M}$. $\mathfrak{M}$ $\models_{prop}$ *?*φ **by** *fastforce*
hence ⊢ ⦇ *?*φ ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
moreover
have *map snd* ($\mathfrak{V}$ Δ) = Δ **by** (*induct* Δ, *auto*)
hence ⊢ $\bigsqcup$ (*map* $\bigsqcap$ *?*Φ @ *map* $\bigsqcap$ *?*Ψ) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?*Φ) ⊔ $\bigsqcup$ (*map*

$\bigsqcap$ *?Ψ*))
$\vdash \bigsqcup$ (*map* $\bigsqcap$ *?Δ* @ *map* $\bigsqcap$ *?Ω*) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?Δ*) ⊔ $\bigsqcup$ (*map* $\bigsqcap$ *?Ω*))
$\vdash \bigsqcup$ (*map* $\bigsqcap$ *?Φ*) ↔ ($\bigsqcap$ [*δ*] ⊓ $\bigsqcup$ (*map* $\bigsqcap$ *?Σ*$_\alpha$))
$\vdash \bigsqcup$ (*map* $\bigsqcap$ *?Ψ*) ↔ ($\bigsqcap$ Δ ⊓ $\bigsqcup$ (*map* $\bigsqcap$ *?Σ*$_\alpha$))
$\vdash \bigsqcup$ (*map* $\bigsqcap$ *?Δ*) ↔ ($\bigsqcap$ [*δ*] ⊓ $\bigsqcup$ (*map* $\bigsqcap$ *?Σ*))
$\vdash \bigsqcup$ (*map* $\bigsqcap$ *?Ω*) ↔ ($\bigsqcap$ Δ ⊓ $\bigsqcup$ (*map* $\bigsqcap$ *?Σ*))
using *arbitrary-disj-concat-equiv*
extract-inner-concat [**where** Δ = [(Δ, *δ*)] **and** Ψ = *?T*$_\Sigma$ Ψ]
extract-inner-concat [**where** Δ = 𝔙 Δ **and** Ψ = *?T*$_\Sigma$ Ψ]
extract-inner-concat-remdups [**where** Δ = [(Δ, *δ*)] **and** Ψ = *?T*$_\Sigma$ Ψ]
extract-inner-concat-remdups [**where** Δ = 𝔙 Δ **and** Ψ = *?T*$_\Sigma$ Ψ]
by *auto*
ultimately have
$\vdash$ (($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*$_\alpha$) → *φ*) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*) → *φ*)) →
($\bigsqcup$ (*map* $\bigsqcap$ *?Φ* @ *map* $\bigsqcap$ *?Ψ*) → *φ*) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?Δ* @ *map* $\bigsqcap$ *?Ω*) → *φ*)
using *modus-ponens* **by** *blast*
moreover have (#) (Δ, *δ*) = (@) [(Δ, *δ*)] **by** *fastforce*
ultimately have
$\vdash$ (($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*$_\alpha$) → *φ*) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*) → *φ*)) →
(($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*$_\alpha$′) → *φ*) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*′) → *φ*))
by *auto*
}
hence
$\vdash$ (($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*$_\alpha$′) → *φ*) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*′) → *φ*))
using *Cons modus-ponens* **by** *blast*
moreover have Δ*δ* = (*?Δ*,*?δ*) **by** *fastforce*
ultimately show *?case* **by** *metis*
qed
}
hence $\vdash$ ($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*$_\alpha$) → *φ*) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*) → *φ*) **by** *blast*
moreover have $\vdash$ (*?Ψ*′ :→ *φ*) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ *?Σ*$_\alpha$) → *φ*)
proof (*induct* Ψ$_0$)
case *Nil*
have $\vdash$ *φ* ↔ ((⊤ ⊔ ⊥) → *φ*)
proof −
let *?φ* = ⟨*φ*⟩ ↔ ((⊤ ⊔ ⊥) → ⟨*φ*⟩)
have ∀ 𝔐. 𝔐 ⊨$_{prop}$ *?φ* **by** *fastforce*
hence $\vdash$ ⦇ *?φ* ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
thus *?case* **by** *simp*
next
case (*Cons* *ψ*$_0$ Ψ$_0$)
let *?Ξ* = *fst* *ψ*$_0$
let *?δ* = *snd* *ψ*$_0$
let *?Ψ*′ = *map* (λ(Ψ,*ψ*). (Ψ :→ *φ* → *ψ*)) Ψ$_0$

let $?\Sigma_\alpha$ = *map* (*map snd*) ($?T_\Sigma$ Ψ_0)
{
fix Ξ :: *'a list*
have *map snd* ($\mathfrak{V}$ Ξ) = Ξ **by** (*induct* Ξ, *auto*)
hence *map snd* ∘ (@) ($\mathfrak{V}$ Ξ) = (@) Ξ ∘ *map snd* **by** *fastforce*
}
moreover have (*map snd* ∘ (#) (*?*Ξ, *?*δ)) = (@) [*?*δ] ∘ *map snd* **by** *fastforce*
ultimately have †:
map (*map snd*) ($?T_\Sigma$ (ψ_0 # Ψ_0)) = *map* ((#) *?*δ) $?\Sigma_\alpha$ @ *map* ((@) *?*Ξ) $?\Sigma_\alpha$
map ($\lambda(\Psi,\psi)$. (Ψ :⇢ φ → ψ)) (ψ_0 # Ψ_0) = *?*Ξ :⇢ φ → *?*δ # $?\Psi'$
by (*simp add*: *case-prod-beta'*)+
have *A*: ⊢ ($?\Psi'$:⇢ φ) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ $?\Sigma_\alpha$) → φ) **using** *Cons.hyps* **by** *auto*
have *B*: ⊢ (*?*Ξ :⇢ φ) ↔ ($\bigsqcap$ *?*Ξ → φ)
by (*simp add*: *list-curry-uncurry*)
have *C*: ⊢ $\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$) @ *map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$))
↔ ($\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$)) ⊔ $\bigsqcup$ (*map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$)))
using *arbitrary-disj-concat-equiv* **by** *blast*
have *map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$) = (*map* ((⊓) *?*δ) (*map* $\bigsqcap$ $?\Sigma_\alpha$)) **by** *auto*
hence *D*: ⊢ $\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$)) ↔ (*?*δ ⊓ $\bigsqcup$ (*map* $\bigsqcap$ $?\Sigma_\alpha$))
using *conj-extract* **by** *presburger*
have *E*: ⊢ $\bigsqcup$ (*map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$)) ↔ ($\bigsqcap$ *?*Ξ ⊓ $\bigsqcup$ (*map* $\bigsqcap$ $?\Sigma_\alpha$))
using *conj-multi-extract* **by** *blast*
have
⊢ ($?\Psi'$:⇢ φ) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ $?\Sigma_\alpha$) → φ)
→ (*?*Ξ :⇢ φ) ↔ ($\bigsqcap$ *?*Ξ → φ)
→ $\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$) @ *map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$))
↔ ($\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$)) ⊔ $\bigsqcup$ (*map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$)))
→ $\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$)) ↔ (*?*δ ⊓ $\bigsqcup$ (*map* $\bigsqcap$ $?\Sigma_\alpha$))
→ $\bigsqcup$ (*map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$)) ↔ ($\bigsqcap$ *?*Ξ ⊓ $\bigsqcup$ (*map* $\bigsqcap$ $?\Sigma_\alpha$))
→ ((*?*Ξ :⇢ φ → *?*δ) → $?\Psi'$:⇢ φ)
↔ ($\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$) @ *map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$)) → φ)
proof −
let $?\varphi$ =
⟨$?\Psi'$:⇢ φ⟩ ↔ (⟨$\bigsqcup$ (*map* $\bigsqcap$ $?\Sigma_\alpha$)⟩ → ⟨φ⟩)
→ ⟨(*?*Ξ :⇢ φ)⟩ ↔ (⟨$\bigsqcap$ *?*Ξ⟩ → ⟨φ⟩)
→ ⟨$\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$) @ *map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$))⟩
↔ (⟨$\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$))⟩ ⊔ ⟨$\bigsqcup$ (*map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$))⟩)
→ ⟨$\bigsqcup$ (*map* $\bigsqcap$ (*map* ((#) *?*δ) $?\Sigma_\alpha$))⟩ ↔ (⟨*?*δ⟩ ⊓ ⟨$\bigsqcup$ (*map* $\bigsqcap$ $?\Sigma_\alpha$)⟩)
→ ⟨$\bigsqcup$ (*map* $\bigsqcap$ (*map* ((@) *?*Ξ) $?\Sigma_\alpha$))⟩ ↔ (⟨$\bigsqcap$ *?*Ξ⟩ ⊓ ⟨$\bigsqcup$ (*map* $\bigsqcap$ $?\Sigma_\alpha$)⟩)
→ ((⟨*?*Ξ :⇢ φ⟩ → ⟨*?*δ⟩) → ⟨$?\Psi'$:⇢ φ⟩)

```
                  ↔ (⟨⨆ (map ⨅ (map ((#) ?δ) ?Σα) @ map ⨅ (map ((@) ?Ξ)
?Σα))⟩ → ⟨φ⟩)
        have ∀𝔐. 𝔐 ⊨prop ?φ by fastforce
        hence ⊢ ⦇ ?φ ⦈ using propositional-semantics by blast
        thus ?thesis by simp
      qed
      hence
        ⊢    ((?Ξ :→ φ → ?δ) → ?Ψ' :→ φ)
          ↔ (⨆ (map ⨅ (map ((#) ?δ) ?Σα) @ map ⨅ (map ((@) ?Ξ) ?Σα)) →
φ)
        using A B C D E modus-ponens by blast
      thus ?case using † by simp
    qed
    ultimately show ?thesis using biconditional-transitivity-rule Ψ0 by blast
  qed
  have II: ∀ σ ∈ set ?Σ. length σ + 1 ≥ length Ψ
  proof −
    let ?F = length ∘ fst
    let ?S = sort-key (− ?F)
    let ?Σ' = map (map snd ∘ remdups) (?TΣ (?S Ψ0))
    have mset Ψ0 = mset (?S Ψ0) by simp

    have ∀ Φ. mset Ψ0 = mset Φ ⟶ mset (map mset (?TΣ Ψ0)) = mset (map
mset (?TΣ Φ))
    proof (induct Ψ0)
      case Nil
      then show ?case by simp
    next
      case (Cons ψ Ψ0)
      obtain Δ δ where ψ = (Δ,δ) by fastforce
      {
        fix Φ
        assume mset (ψ # Ψ0) = mset Φ
        hence mset Ψ0 = mset (remove1 ψ Φ)
          by (simp add: union-single-eq-diff)
        have ψ ∈ set Φ using ‹mset (ψ # Ψ0) = mset Φ›
          by (metis list.set-intros(1) set-mset-mset)
       hence mset (map mset (?TΣ Φ)) = mset (map mset (?TΣ (ψ # (remove1
ψ Φ))))
        proof (induct Φ)
          case Nil
          then show ?case by simp
        next
          case (Cons φ Φ)
          then show ?case proof (cases φ = ψ)
            case True
            then show ?thesis by simp
          next
            case False
```

let $?\Sigma' = ?T_\Sigma\ (\psi\ \#\ (remove1\ \psi\ \Phi))$
have †: $mset\ (map\ mset\ ?\Sigma') = mset\ (map\ mset\ (?T_\Sigma\ \Phi))$
using *Cons False* **by** *simp*
obtain $\Delta'\ \delta'$
where $\varphi = (\Delta',\delta')$
by *fastforce*
let $?\Sigma = ?T_\Sigma\ (remove1\ \psi\ \Phi)$
let $?\mathfrak{m} = image\text{-}mset\ mset$
have
$mset\ (map\ mset\ (?T_\Sigma\ (\psi\ \#\ remove1\ \psi\ (\varphi\ \#\ \Phi)))) =$
$mset\ (map\ mset\ (?\Pi_C\ \psi\ (?\Pi_C\ \varphi\ ?\Sigma)))$
using *False* **by** *simp*
hence $mset\ (map\ mset\ (?T_\Sigma\ (\psi\ \#\ remove1\ \psi\ (\varphi\ \#\ \Phi)))) =$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((\#)\ \psi) \circ image\text{-}mset\ ((\#)\ \varphi)))\ (mset\ ?\Sigma) +$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((\#)\ \psi) \circ image\text{-}mset\ ((@)\ (\mathfrak{V}\ \Delta'))))\ (mset\ ?\Sigma) +$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((@)\ (\mathfrak{V}\ \Delta)) \circ image\text{-}mset\ ((\#)\ \varphi)))\ (mset\ ?\Sigma) +$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((@)\ (\mathfrak{V}\ \Delta)) \circ image\text{-}mset\ ((@)\ (\mathfrak{V}\ \Delta'))))\ (mset\ ?\Sigma)$
using $\langle\psi = (\Delta,\delta)\rangle\ \langle\varphi = (\Delta',\delta')\rangle$
by (*simp add*: *multiset.map-comp*)
hence $mset\ (map\ mset\ (?T_\Sigma\ (\psi\ \#\ remove1\ \psi\ (\varphi\ \#\ \Phi)))) =$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((\#)\ \varphi) \circ image\text{-}mset\ ((\#)\ \psi)))\ (mset\ ?\Sigma) +$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((@)\ (\mathfrak{V}\ \Delta')) \circ image\text{-}mset\ ((\#)\ \psi)))\ (mset\ ?\Sigma) +$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((\#)\ \varphi) \circ image\text{-}mset\ ((@)\ (\mathfrak{V}\ \Delta))))\ (mset\ ?\Sigma) +$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((@)\ (\mathfrak{V}\ \Delta')) \circ image\text{-}mset\ ((@)\ (\mathfrak{V}\ \Delta))))\ (mset\ ?\Sigma)$
by (*simp add*: *image-mset-cons-homomorphism*
image-mset-append-homomorphism
image-mset-add-collapse
add-mset-commute
add.commute)
hence $mset\ (map\ mset\ (?T_\Sigma\ (\psi\ \#\ remove1\ \psi\ (\varphi\ \#\ \Phi)))) =$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((\#)\ \varphi)))\ (mset\ ?\Sigma') +$
$(?\mathfrak{m} \circ (image\text{-}mset\ ((@)\ (\mathfrak{V}\ \Delta'))))\ (mset\ ?\Sigma')$
using $\langle\psi = (\Delta,\delta)\rangle$
by (*simp add*: *multiset.map-comp*)
hence $mset\ (map\ mset\ (?T_\Sigma\ (\psi\ \#\ remove1\ \psi\ (\varphi\ \#\ \Phi)))) =$
$image\text{-}mset\ ((+)\ \{\#\varphi\#\})\ (mset\ (map\ mset\ ?\Sigma')) +$
$image\text{-}mset\ ((+)\ (mset\ (\mathfrak{V}\ \Delta')))\ (mset\ (map\ mset\ ?\Sigma'))$
by (*simp add*: *image-mset-cons-homomorphism*
image-mset-append-homomorphism)
hence $mset\ (map\ mset\ (?T_\Sigma\ (\psi\ \#\ remove1\ \psi\ (\varphi\ \#\ \Phi)))) =$
$image\text{-}mset\ ((+)\ \{\#\varphi\#\})\ (mset\ (map\ mset\ (?T_\Sigma\ \Phi))) +$
$image\text{-}mset\ ((+)\ (mset\ (\mathfrak{V}\ \Delta')))\ (mset\ (map\ mset\ (?T_\Sigma\ \Phi)))$
using † **by** *auto*

```
          hence mset (map mset (?T_Σ (ψ # remove1 ψ (φ # Φ)))) =
                (?m ∘ (image-mset ((#) φ))) (mset (?T_Σ Φ)) +
                (?m ∘ (image-mset ((@) (𝔙 Δ')))) (mset (?T_Σ Φ))
            by (simp add: image-mset-cons-homomorphism
                          image-mset-append-homomorphism)
          thus ?thesis using ‹φ = (Δ',δ')› by (simp add: multiset.map-comp)
        qed
      qed
       hence   image-mset mset (image-mset ((#) ψ) (mset (?T_Σ (remove1 ψ
Φ)))) +
               image-mset mset (image-mset ((@) (𝔙 Δ)) (mset (?T_Σ (remove1
ψ Φ))))
             = image-mset mset (mset (?T_Σ Φ))
        by (simp add: ‹ψ = (Δ,δ)› multiset.map-comp)
      hence
         image-mset ((+) {# ψ #}) (image-mset mset (mset (?T_Σ (remove1 ψ
Φ)))) +
        image-mset ((+) (mset (𝔙 Δ))) (image-mset mset (mset (?T_Σ (remove1
ψ Φ))))
         = image-mset mset (mset (?T_Σ Φ))
      by (simp add: image-mset-cons-homomorphism image-mset-append-homomorphism)
       hence
        image-mset ((+) {# ψ #}) (image-mset mset (mset (?T_Σ Ψ_0))) +
         image-mset ((+) (mset (𝔙 Δ))) (image-mset mset (mset (?T_Σ Ψ_0)))
       = image-mset mset (mset (?T_Σ Φ))
        using Cons ‹mset Ψ_0 = mset (remove1 ψ Φ)›
        by fastforce
       hence
        image-mset mset (image-mset ((#) ψ) (mset (?T_Σ Ψ_0))) +
         image-mset mset (image-mset ((@) (𝔙 Δ)) (mset (?T_Σ Ψ_0)))
       = image-mset mset (mset (?T_Σ Φ))
      by (simp add: image-mset-cons-homomorphism image-mset-append-homomorphism)
       hence mset (map mset (?T_Σ (ψ # Ψ_0))) = mset (map mset (?T_Σ Φ))
        by (simp add: ‹ψ = (Δ,δ)› multiset.map-comp)
      }
      then show ?case by blast
    qed
    hence mset (map mset (?T_Σ Ψ_0)) = mset (map mset (?T_Σ (?S Ψ_0)))
      using ‹mset Ψ_0 = mset (?S Ψ_0)› by blast
    hence    mset (map (mset ∘ (map snd) ∘ remdups) (?T_Σ Ψ_0))
           = mset (map (mset ∘ (map snd) ∘ remdups) (?T_Σ (?S Ψ_0)))
      using mset-mset-map-snd-remdups by blast
    hence mset (map mset ?Σ) = mset (map mset ?Σ')
      by (simp add: fun.map-comp)
    hence set (map mset ?Σ) = set (map mset ?Σ')
      using mset-eq-setD by blast
    hence ∀ σ ∈ set ?Σ. ∃ σ' ∈ set ?Σ'. mset σ = mset σ'
      by fastforce
    hence ∀ σ ∈ set ?Σ. ∃ σ' ∈ set ?Σ'. length σ = length σ'
```

```
    using mset-eq-length by blast
  have mset (?S Ψ0) ⊆# mset (𝔘 Ξ)
    by (simp add: Ψ0(1))
  {
    fix n
    have ∀ Ψ. mset Ψ ⊆# mset (𝔘 Ξ) ⟶
              sorted (map (− ?F) Ψ) ⟶
              length Ψ = n ⟶
              (∀ σ' ∈ set (map (map snd ∘ remdups) (?T_Σ Ψ)). length σ' + 1
≥ n)
    proof (induct n)
      case 0
      then show ?case by simp
    next
      case (Suc n)
      {
        fix Ψ :: ('a list × 'a) list
        assume A: mset Ψ ⊆# mset (𝔘 Ξ)
           and B: sorted (map (− ?F) Ψ)
           and C: length Ψ = n + 1
        obtain Δ δ where (Δ, δ) = hd Ψ
          using prod.collapse by blast
        let ?Ψ' = tl Ψ
        have mset ?Ψ' ⊆# mset (𝔘 Ξ) using A
        by (induct Ψ, simp, simp, meson mset-subset-eq-insertD subset-mset-def)
        moreover
        have sorted (map (− ?F) (tl Ψ))
          using B
          by (simp add: map-tl sorted-tl)
        moreover have length ?Ψ' = n using C
          by simp
        ultimately have ⋆: ∀ σ' ∈ set (map (map snd ∘ remdups) (?T_Σ ?Ψ')).
length σ' + 1 ≥ n
          using Suc
          by blast
        from C have Ψ = (Δ, δ) # ?Ψ'
          by (metis ‹(Δ, δ) = hd Ψ›
                    One-nat-def
                    add-is-0
                    list.exhaust-sel
                    list.size(3)
                    nat.simps(3))
        have distinct ((Δ, δ) # ?Ψ')
          using A ‹Ψ = (Δ, δ) # ?Ψ'›
                MaxSAT-optimal-pre-witness-distinct
                mset-distinct-msub-down
          by fastforce
        hence set ((Δ, δ) # ?Ψ') ⊆ set (𝔘 Ξ)
          by (metis A ‹Ψ = (Δ, δ) # ?Ψ'›
```

```
        Un-iff
        mset-le-perm-append
        perm-set-eq set-append
        subsetI)
  hence ∀ (Δ′, δ′) ∈ set ?Ψ′. (Δ, δ) ≠ (Δ′, δ′)
      ∀ (Δ′, δ′) ∈ set (𝔘 Ξ). ((Δ, δ) ≠ (Δ′, δ′)) ⟶ (length Δ ≠ length
Δ′)
      set ?Ψ′ ⊆ set (𝔘 Ξ)
    using MaxSAT-optimal-pre-witness-length-iff-eq [where Ψ=Ξ]
      ‹distinct ((Δ, δ) # ?Ψ′)›
    by auto
  hence ∀ (Δ′, δ′) ∈ set ?Ψ′. length Δ ≠ length Δ′
      sorted (map (− ?F) ((Δ, δ) # ?Ψ′))
    using B ‹Ψ = (Δ, δ) # ?Ψ′›
    by (fastforce, auto)
  hence ∀ (Δ′, δ′) ∈ set ?Ψ′. length Δ > length Δ′
    by fastforce
  {
    fix σ′ :: 'a list
    assume σ′ ∈ set (map (map snd ∘ remdups) (?T_Σ Ψ))
    hence σ′ ∈ set (map (map snd ∘ remdups) (?T_Σ ((Δ, δ) # ?Ψ′)))
      using ‹Ψ = (Δ, δ) # ?Ψ′›
      by simp
    from this obtain ψ where ψ:
      ψ ∈ set (?T_Σ ?Ψ′)
      σ′ = (map snd ∘ remdups ∘ (#) (Δ, δ)) ψ ∨
       σ′ = (map snd ∘ remdups ∘ (@) (𝔘 Δ)) ψ
      by fastforce
    hence length σ′ ≥ n
    proof (cases σ′ = (map snd ∘ remdups ∘ (#) (Δ, δ)) ψ)
      case True
      {
        fix Ψ :: ('a list × 'a) list
        fix n :: nat
        assume ∀ (Δ, δ) ∈ set Ψ. n > length Δ
        hence ∀ σ ∈ set (?T_Σ Ψ). ∀ (Δ, δ) ∈ set σ. n > length Δ
        proof (induct Ψ)
          case Nil
          then show ?case by simp
        next
          case (Cons ψ Ψ)
          obtain Δ δ where ψ = (Δ, δ)
            by fastforce
          hence n > length Δ using Cons.prems by fastforce
          have 0: ∀ σ ∈ set (?T_Σ Ψ). ∀ (Δ′, δ′) ∈ set σ. n > length Δ′
            using Cons by simp
          {
            fix σ :: ('a list × 'a) list
            fix ψ′ :: 'a list × 'a
```

```
                assume 1: σ ∈ set (?T_Σ (ψ # Ψ))
                  and 2: ψ' ∈ set σ
                obtain Δ' δ' where ψ' = (Δ', δ')
                  by fastforce
                 have 3: σ ∈ (#) (Δ, δ) ‘ set (?T_Σ Ψ) ∨ σ ∈ (@) (𝔘 Δ) ‘ set
(?T_Σ Ψ)
                  using 1 ‹ψ = (Δ, δ)› by simp
                have n > length Δ'
                proof (cases σ ∈ (#) (Δ, δ) ‘ set (?T_Σ Ψ))
                  case True
                  from this obtain σ' where
                    set σ = insert (Δ, δ) (set σ')
                    σ' ∈ set (?T_Σ Ψ)
                    by auto
                  then show ?thesis
                    using 0 ‹ψ' ∈ set σ› ‹ψ' = (Δ', δ')› ‹n > length Δ›
                    by auto
                next
                  case False
                  from this and 3 obtain σ' where σ':
                    set σ = set (𝔘 Δ) ∪ (set σ')
                    σ' ∈ set (?T_Σ Ψ)
                    by auto
                  have ∀ (Δ', δ') ∈ set (𝔘 Δ). length Δ > length Δ'
                    by (metis (mono-tags, lifting)
                              case-prodI2
                              MaxSAT-optimal-pre-witness-nonelement
                              not-le)
                  hence ∀ (Δ', δ') ∈ set (𝔘 Δ). n > length Δ'
                    using ‹n > length Δ› by auto
                 then show ?thesis using 0 σ' ‹ψ' ∈ set σ› ‹ψ' = (Δ', δ')› by
fastforce
                  qed
                  hence n > length (fst ψ') using ‹ψ' = (Δ', δ')› by fastforce
                }
                then show ?case by fastforce
              qed
             }
            hence ∀ σ ∈ set (?T_Σ ?Ψ'). ∀ (Δ', δ') ∈ set σ. length Δ > length Δ'
              using ‹∀ (Δ', δ') ∈ set ?Ψ'. length Δ > length Δ'›
              by blast
             then show ?thesis using True ⋆ ψ(1) by fastforce
            next
             case False
             have ∀ (Δ', δ') ∈ set ?Ψ'. length Δ ≥ length Δ'
              using ‹∀ (Δ', δ') ∈ set ?Ψ'. length Δ > length Δ'›
              by auto
             hence ∀ (Δ', δ') ∈ set Ψ. length Δ ≥ length Δ'
              using ‹Ψ = (Δ, δ) # ?Ψ'›
```

```
            by (metis case-prodI2 eq-iff prod.sel(1) set-ConsD)
          hence length Δ + 1 ≥ length Ψ
            using A MaxSAT-optimal-pre-witness-pigeon-hole
            by fastforce
          hence length Δ ≥ n
            using C
            by simp
          have length Δ = length (𝔙 Δ)
            by (induct Δ, simp+)
          hence length (remdups (𝔙 Δ)) = length (𝔙 Δ)
            by (simp add: MaxSAT-optimal-pre-witness-distinct)
          hence length (remdups (𝔙 Δ)) ≥ n
            using ‹length Δ = length (𝔙 Δ)› ‹n ≤ length Δ›
            by linarith
          have mset (remdups (𝔙 Δ @ ψ)) = mset (remdups (ψ @ 𝔙 Δ))
            by (simp add: mset-remdups)
          hence length (remdups (𝔙 Δ @ ψ)) ≥ length (remdups (𝔙 Δ))
                by (metis le-cases length-sub-mset mset-remdups-append-msub
size-mset)
          hence length (remdups (𝔙 Δ @ ψ)) ≥ n
            using ‹n ≤ length (remdups (𝔙 Δ))› dual-order.trans by blast
          thus ?thesis using False ψ(2)
            by simp
        qed
       }
       hence ∀ σ' ∈ set (map (map snd ∘ remdups) (?T_Σ Ψ)). length σ' ≥ n
        by blast
      }
      then show ?case by fastforce
    qed
   }
   hence ∀ σ' ∈ set ?Σ'. length σ' + 1 ≥ length (?𝒮 Ψ₀)
    using ‹mset (?𝒮 Ψ₀) ⊆# mset (𝔙 Ξ)›
    by fastforce
   hence ∀ σ' ∈ set ?Σ'. length σ' + 1 ≥ length Ψ₀ by simp
   hence ∀ σ ∈ set ?Σ. length σ + 1 ≥ length Ψ₀
    using ‹∀ σ ∈ set ?Σ. ∃ σ' ∈ set ?Σ'. length σ = length σ'›
    by fastforce
   thus ?thesis using Ψ₀ by fastforce
  qed
  have III: ∀ σ ∈ set ?Σ. mset σ ⊆# mset Ξ
  proof −
   have remdups (𝔙 Ξ) = 𝔙 Ξ
    by (simp add: MaxSAT-optimal-pre-witness-distinct distinct-remdups-id)
   from Ψ₀(1) have set Ψ₀ ⊆ set (𝔙 Ξ)
    by (metis (no-types, lifting) ‹remdups (𝔙 Ξ) = 𝔙 Ξ›
                                  mset-remdups-set-sub-iff
                                  mset-remdups-subset-eq
                                  subset-mset.dual-order.trans)
```

```
    hence ∀ σ ∈ set (?T_Σ Ψ_0). set σ ⊆ set (𝔙 Ξ)
    proof (induct Ψ_0)
      case Nil
      then show ?case by simp
    next
      case (Cons ψ Ψ_0)
      hence ∀ σ ∈ set (?T_Σ Ψ_0). set σ ⊆ set (𝔙 Ξ) by auto
      obtain Δ δ where ψ = (Δ,δ) by fastforce
      hence (Δ, δ) ∈ set (𝔙 Ξ) using Cons by simp
      {
        fix σ :: ('a list × 'a) list
        assume ⋆: σ ∈ (#) (Δ, δ) ‘ set (?T_Σ Ψ_0) ∪ (@) (𝔙 Δ) ‘ set (?T_Σ Ψ_0)
        have set σ ⊆ set (𝔙 Ξ)
        proof (cases σ ∈ (#) (Δ, δ) ‘ set (?T_Σ Ψ_0))
          case True
          then show ?thesis
            using ‹∀ σ ∈ set (?T_Σ Ψ_0). set σ ⊆ set (𝔙 Ξ)› ‹(Δ, δ) ∈ set (𝔙 Ξ)›
            by fastforce
        next
          case False
          hence σ ∈ (@) (𝔙 Δ) ‘ set (?T_Σ Ψ_0) using ⋆ by simp
          moreover have set (𝔙 Δ) ⊆ set (𝔙 Ξ)
           using MaxSAT-optimal-pre-witness-element-inclusion ‹(Δ, δ) ∈ set (𝔙
Ξ)›
            by fastforce
          ultimately show ?thesis
            using ‹∀ σ ∈ set (?T_Σ Ψ_0). set σ ⊆ set (𝔙 Ξ)›
            by force
        qed
      }
      hence ∀σ∈(#) (Δ, δ) ‘ set (?T_Σ Ψ_0) ∪ (@) (𝔙 Δ) ‘ set (?T_Σ Ψ_0). set σ
⊆ set (𝔙 Ξ)
        by auto
      thus ?case using ‹ψ = (Δ, δ)› by simp
    qed
    hence ∀ σ ∈ set (?T_Σ Ψ_0). mset (remdups σ) ⊆# mset (remdups (𝔙 Ξ))
      using mset-remdups-set-sub-iff by blast
    hence ∀ σ ∈ set ?Σ. mset σ ⊆# mset (map snd (𝔙 Ξ))
      using map-monotonic ‹remdups (𝔙 Ξ) = 𝔙 Ξ›
      by auto
    moreover have map snd (𝔙 Ξ) = Ξ by (induct Ξ, simp+)
    ultimately show ?thesis by simp
  qed
  show ?thesis using I II III by fastforce
qed
from this obtain Σ_0 where Σ_0:
  ⊢ (Ψ :→ φ) ↔ (⋃ (map ⋂ Σ_0) → φ)
  ∀ σ ∈ set Σ_0. mset σ ⊆# mset Ξ ∧ length σ + 1 ≥ length Ψ
  by blast
```

```
  moreover
  have (Φ @ Ψ) :→ φ = Φ :→ (Ψ :→ φ) by (induct Φ, simp+)
  hence ⊢ ((Φ @ Ψ) :→ φ) ↔ (⨅ Φ → (Ψ :→ φ))
    by (simp add: list-curry-uncurry)
  moreover have ⊢ (Ψ :→ φ) ↔ (⨆ (map ⨅ Σ₀) → φ)
              → (Φ @ Ψ) :→ φ ↔ (⨅ Φ → Ψ :→ φ)
              → (Φ @ Ψ) :→ φ ↔ ((⨅ Φ ⊓ ⨆ (map ⨅ Σ₀)) → φ)
  proof −
    let ?φ = ⟨Ψ :→ φ⟩ ↔ (⟨⨆ (map ⨅ Σ₀)⟩ → ⟨φ⟩)
          → ⟨(Φ @ Ψ) :→ φ⟩ ↔ (⟨⨅ Φ⟩ → ⟨Ψ :→ φ⟩)
          → ⟨(Φ @ Ψ) :→ φ⟩ ↔ ((⟨⨅ Φ⟩ ⊓ ⟨⨆ (map ⨅ Σ₀)⟩) → ⟨φ⟩)
    have ∀ 𝔐. 𝔐 ⊨_prop ?φ by fastforce
    hence ⊢ ⦇ ?φ ⦈ using propositional-semantics by blast
    thus ?thesis by simp
  qed
  moreover
  let ?Σ = map ((@) Φ) Σ₀
  have ∀ φ ψ χ. ⊢ (φ → ψ) → χ → ψ ∨ ¬ ⊢ χ → φ
    by (meson modus-ponens flip-hypothetical-syllogism)
  hence ⊢ ((⨅ Φ ⊓ ⨆ (map ⨅ Σ₀)) → φ) ↔ (⨆ (map ⨅ ?Σ) → φ)
    using append-dnf-distribute biconditional-def by fastforce
  ultimately have ⊢ (Φ @ Ψ) :→ φ ↔ (⨆ (map ⨅ ?Σ) → φ)
    using modus-ponens biconditional-transitivity-rule
    by blast
  moreover
  {
    fix σ
    assume σ ∈ set ?Σ
    from this obtain σ₀ where σ₀: σ = Φ @ σ₀ σ₀ ∈ set Σ₀ by (simp, blast)
    hence mset σ₀ ⊆# mset Ξ using Σ₀(2) by blast
    hence mset σ ⊆# mset (Φ @ Ξ) using σ₀(1) by simp
    hence mset σ ⊆# mset Γ using assms(1) assms(2)
      by (simp, meson subset-mset.dual-order.trans subset-mset.le-diff-conv2)
    moreover
    have length σ + 1 ≥ length (Φ @ Ψ) using Σ₀(2) σ₀ by simp
    ultimately have mset σ ⊆# mset Γ length σ + 1 ≥ length (Φ @ Ψ) by auto
  }
  ultimately
  show ?thesis by blast
qed

lemma (in classical-logic) relative-maximals-optimal-witness:
  assumes ¬ ⊢ φ
  shows 0 < (‖ Γ ‖_φ)
    = (∃ Σ. mset (map snd Σ) ⊆# mset Γ ∧
          map (uncurry (⊔)) Σ :⊢ φ ∧
          1 + (‖ map (uncurry (→)) Σ @ Γ ⊖ map snd Σ ‖_φ) = ‖ Γ ‖_φ)
proof (rule iffI)
  assume 0 < ‖ Γ ‖_φ
```

from *this* **obtain** Ξ **where** Ξ: $\Xi \in \mathcal{M}\ \Gamma\ \varphi\ \mathit{length}\ \Xi < \mathit{length}\ \Gamma$
using ‹$\neg \vdash \varphi$›
complement-relative-MaxSAT-def
relative-MaxSAT-intro
relative-maximals-existence
by *fastforce*
from *this* **obtain** ψ **where** ψ: $\psi \in \mathit{set}\ (\Gamma \ominus \Xi)$
by (*metis* ‹$0 < \| \Gamma \|_{\varphi}$›
less-not-refl
list.exhaust
list.set-intros(*1*)
list.size(*3*)
complement-relative-MaxSAT-intro)
let $?\Sigma = \mathfrak{W}\ \varphi\ (\psi\ \#\ \Xi)$
let $?\Sigma_A = \mathfrak{W}_{\sqcup}\ \varphi\ (\psi\ \#\ \Xi)$
let $?\Sigma_B = \mathfrak{W}_{\rightarrow}\ \varphi\ (\psi\ \#\ \Xi)$
have $\diamondsuit$: $\mathit{mset}\ (\psi\ \#\ \Xi) \subseteq\#\ \mathit{mset}\ \Gamma$
$\psi\ \#\ \Xi :\vdash \varphi$
using $\Xi(1)\ \psi$
relative-maximals-def
list-deduction-theorem
relative-maximals-complement-deduction
msub-list-subtract-elem-cons-msub [**where** Ξ=Ξ]
by *blast*+
moreover have $\mathit{map}\ \mathit{snd}\ ?\Sigma = \psi\ \#\ \Xi$ **by** (*induct* Ξ, *simp*+)
ultimately have $?\Sigma_A :\vdash \varphi$
$\mathit{mset}\ (\mathit{map}\ \mathit{snd}\ ?\Sigma) \subseteq\#\ \mathit{mset}\ \Gamma$
using *MaxSAT-optimal-witness-deduction*
list-deduction-def weak-biconditional-weaken
by (*metis*+)
moreover
{
let $?\Gamma' = ?\Sigma_B\ @\ \Gamma \ominus \mathit{map}\ \mathit{snd}\ ?\Sigma$
have *A*: $\mathit{length}\ ?\Sigma_B = 1 + \mathit{length}\ \Xi$
by (*induct* Ξ, *simp*+)
have *B*: $?\Sigma_B \in \mathcal{M}\ ?\Gamma'\ \varphi$
proof −
have $\neg\ ?\Sigma_B :\vdash \varphi$
by (*metis* (*no-types*, *lifting*)
$\Xi(1)$ ‹$?\Sigma_A :\vdash \varphi$›
modus-ponens list-deduction-def
optimal-witness-split-identity
relative-maximals-def
mem-Collect-eq)
moreover have $\mathit{mset}\ ?\Sigma_B \subseteq\#\ \mathit{mset}\ ?\Gamma'$
by *simp*
hence $\forall\ \Psi.\ \mathit{mset}\ \Psi \subseteq\#\ \mathit{mset}\ ?\Gamma' \longrightarrow \neg\ \Psi :\vdash \varphi \longrightarrow \mathit{length}\ \Psi \leq \mathit{length}\ ?\Sigma_B$
proof −
have $\forall\ \Psi \in \mathcal{M}\ ?\Gamma'\ \varphi.\ \mathit{length}\ \Psi = \mathit{length}\ ?\Sigma_B$

proof (*rule ccontr*)
assume $\neg$ ($\forall$ $\Psi \in \mathcal{M}$ $?\Gamma'$ φ. *length* Ψ = *length* $?\Sigma_B$)
from *this* **obtain** Ψ **where**
Ψ: $\Psi \in \mathcal{M}$ $?\Gamma'$ φ
length $\Psi \neq$ *length* $?\Sigma_B$
by *blast*
have *length* $\Psi \geq$ *length* $?\Sigma_B$
using Ψ(*1*)
‹$\neg$ $?\Sigma_B$:⊢ φ›
‹*mset* $?\Sigma_B$ ⊆# *mset* $?\Gamma'$›
unfolding *relative-maximals-def*
by *blast*
hence *length* $\Psi >$ *length* $?\Sigma_B$
using Ψ(*2*)
by *linarith*
have *length* Ψ = *length* (Ψ ⊖ $?\Sigma_B$) + *length* (Ψ ∩ $?\Sigma_B$)
(**is** *length* Ψ = *length* $?A$ + *length* $?B$)
by (*metis* (*no-types*, *lifting*)
length-append
list-diff-intersect-comp
mset-append
mset-eq-length)
{
fix σ
assume *mset* σ ⊆# *mset* Γ
length σ + *1* ≥ *length* ($?A$ @ $?B$)
hence *length* σ + *1* ≥ *length* Ψ
using ‹*length* Ψ = *length* $?A$ + *length* $?B$›
by *simp*
hence *length* σ + *1* > *length* $?\Sigma_B$
using ‹*length* $\Psi >$ *length* $?\Sigma_B$› **by** *linarith*
hence *length* σ + *1* > *length* Ξ + *1*
using *A* **by** *simp*
hence *length* $\sigma >$ *length* Ξ **by** *linarith*
have σ :⊢ φ
proof (*rule ccontr*)
assume $\neg$ σ :⊢ φ
hence *length* $\sigma \leq$ *length* Ξ
using ‹*mset* σ ⊆# *mset* Γ› Ξ(*1*)
unfolding *relative-maximals-def*
by *blast*
thus *False* **using** ‹*length* $\sigma >$ *length* Ξ› **by** *linarith*
qed
}
moreover
have *mset* Ψ ⊆# *mset* $?\Gamma'$
$\neg$ Ψ :⊢ φ
$\forall \Phi$. *mset* Φ ⊆# *mset* $?\Gamma' \wedge \neg$ Φ :⊢ $\varphi \longrightarrow$ *length* $\Phi \leq$ *length* Ψ
using Ψ(*1*) *relative-maximals-def* **by** *blast*+

hence *mset ?A* $\subseteq\#$ *mset* ($\Gamma \ominus$ *map snd ?*Σ)
by (*simp add*: *add.commute subset-eq-diff-conv*)
hence *mset ?A* $\subseteq\#$ *mset* ($\Gamma \ominus$ (ψ # Ξ))
using ‹*map snd ?*$\Sigma = \psi$ # Ξ› **by** *metis*
moreover
have *mset ?B* $\subseteq\#$ *mset* ($\mathfrak{W}_{\rightarrow}$ φ (ψ # Ξ))
using *list-intersect-right-project* **by** *blast*
ultimately obtain Σ **where** Σ: $\vdash$ ((*?A* @ *?B*) :→ φ) ↔ ($\bigsqcup$ (*map* $\bigsqcap$ Σ) → φ)
$\forall \sigma \in set\ \Sigma.\ \sigma :\vdash \varphi$
using ♢ *optimal-witness-list-intersect-biconditional*
by *metis*
hence $\vdash \bigsqcup$ (*map* $\bigsqcap$ Σ) → φ
using *weak-disj-of-conj-equiv* **by** *blast*
hence *?A* @ *?B* :$\vdash$ φ
using Σ(*1*) *modus-ponens list-deduction-def weak-biconditional-weaken*
by *blast*
moreover have *set* (*?A* @ *?B*) = *set* Ψ
using *list-diff-intersect-comp union-code set-mset-mset* **by** *metis*
hence *?A* @ *?B* :$\vdash$ φ = Ψ :$\vdash$ φ
using *list-deduction-monotonic* **by** *blast*
ultimately have Ψ :$\vdash$ φ **by** *metis*
thus *False* **using** Ψ(*1*) **unfolding** *relative-maximals-def* **by** *blast*
qed
moreover have $\exists\ \Psi.\ \Psi \in \mathcal{M}$ *?*Γ' φ
using *assms relative-maximals-existence* **by** *blast*
ultimately show *?thesis*
using *relative-maximals-def*
by *fastforce*
qed
ultimately show *?thesis*
unfolding *relative-maximals-def*
by *fastforce*
qed
have *C*: $\forall\ \Xi\ \Gamma\ \varphi.\ \Xi \in \mathcal{M}\ \Gamma\ \varphi \longrightarrow$ *length* Ξ = | Γ |$_\varphi$
using *relative-MaxSAT-intro* **by** *blast*
then have *D*: *length* Ξ = | Γ |$_\varphi$
using ‹$\Xi \in \mathcal{M}\ \Gamma\ \varphi$› **by** *blast*
have
$\forall$ (Σ ::*'a list*) Γ *n*. (¬ *mset* Σ $\subseteq\#$ *mset* Γ ∨ *length* ($\Gamma \ominus \Sigma$) ≠ *n*) ∨ *length* Γ = *n* + *length* Σ
using *list-subtract-msub-eq* **by** *blast*
then have *E*: *length* Γ = *length* ($\Gamma \ominus$ *map snd* ($\mathfrak{W}$ φ (ψ # Ξ))) + *length* (ψ # Ξ)
using ‹*map snd* ($\mathfrak{W}$ φ (ψ # Ξ)) = ψ # Ξ› ‹*mset* (ψ # Ξ) $\subseteq\#$ *mset* Γ› **by** *presburger*
have *1* + *length* Ξ = | $\mathfrak{W}_{\rightarrow}$ φ (ψ # Ξ) @ $\Gamma \ominus$ *map snd* ($\mathfrak{W}$ φ (ψ # Ξ)) |$_\varphi$
using *C B A* **by** *presburger*
hence *1* + (‖ *map* (*uncurry* (→)) *?*Σ @ $\Gamma \ominus$ *map snd ?*Σ ‖$_\varphi$) = ‖ Γ ‖$_\varphi$

```
    using D E ‹map snd (𝔚 φ (ψ # Ξ)) = ψ # Ξ› complement-relative-MaxSAT-def
by force
  }
  ultimately
   show ∃ Σ. mset (map snd Σ) ⊆# mset Γ ∧
             map (uncurry (⊔)) Σ :⊢ φ ∧
             1 + (‖ map (uncurry (→)) Σ @ Γ ⊖ map snd Σ ‖φ) = ‖ Γ ‖φ
  by metis
next
  assume ∃ Σ. mset (map snd Σ) ⊆# mset Γ ∧
              map (uncurry (⊔)) Σ :⊢ φ ∧
              1 + (‖ map (uncurry (→)) Σ @ Γ ⊖ map snd Σ ‖φ) = ‖ Γ ‖φ
  thus 0 < ‖ Γ ‖φ
    by auto
qed


primrec (in implication-logic)
  MaxSAT-witness :: ('a × 'a) list ⇒ 'a list ⇒ ('a × 'a) list (𝔘)
  where
    𝔘 - [] = []
  | 𝔘 Σ (ξ # Ξ) = (case find (λ σ. ξ = snd σ) Σ of
                     None ⇒ 𝔘 Σ Ξ
                   | Some σ ⇒ σ # (𝔘 (remove1 σ Σ) Ξ))

lemma (in implication-logic) MaxSAT-witness-right-msub:
  mset (map snd (𝔘 Σ Ξ)) ⊆# mset Ξ
proof −
  have ∀ Σ. mset (map snd (𝔘 Σ Ξ)) ⊆# mset Ξ
  proof (induct Ξ)
    case Nil
    then show ?case by simp
  next
    case (Cons ξ Ξ)
    {
      fix Σ
      have mset (map snd (𝔘 Σ (ξ # Ξ))) ⊆# mset (ξ # Ξ)
      proof (cases find (λ σ. ξ = snd σ) Σ)
        case None
        then show ?thesis
          by (simp, metis Cons.hyps
                          add-mset-add-single
                          mset-map mset-subset-eq-add-left subset-mset.order-trans)
      next
        case (Some σ)
        note σ = this
        hence ξ = snd σ
          by (meson find-Some-predicate)
        moreover
```

```
      have σ ∈ set Σ
      using σ
      proof (induct Σ)
        case Nil
        then show ?case by simp
      next
        case (Cons σ' Σ)
        then show ?case
          by (cases ξ = snd σ', simp+)
      qed
      ultimately show ?thesis using σ Cons.hyps by simp
    qed
  }
  then show ?case by simp
 qed
 thus ?thesis by simp
qed

lemma (in implication-logic) MaxSAT-witness-left-msub:
  mset (𝔘 Σ Ξ) ⊆# mset Σ
proof −
 have ∀ Σ. mset (𝔘 Σ Ξ) ⊆# mset Σ
 proof (induct Ξ)
   case Nil
   then show ?case by simp
 next
   case (Cons ξ Ξ)
   {
    fix Σ
    have mset (𝔘 Σ (ξ # Ξ)) ⊆# mset Σ
    proof (cases find (λ σ. ξ = snd σ) Σ)
      case None
      then show ?thesis using Cons.hyps by simp
    next
      case (Some σ)
      note σ = this
      hence σ ∈ set Σ
      proof (induct Σ)
        case Nil
        then show ?case by simp
      next
        case (Cons σ' Σ)
        then show ?case
          by (cases ξ = snd σ', simp+)
      qed
       moreover from Cons.hyps have mset (𝔘 (remove1 σ Σ) Ξ) ⊆# mset
(remove1 σ Σ)
        by blast
      hence mset (𝔘 Σ (ξ # Ξ)) ⊆# mset (σ # remove1 σ Σ) using σ by simp
```

```
      ultimately show ?thesis by simp
    qed
  }
  then show ?case by simp
 qed
 thus ?thesis by simp
qed

lemma (in implication-logic) MaxSAT-witness-right-projection:
 mset (map snd (𝔘 Σ Ξ)) = mset ((map snd Σ) ∩ Ξ)
proof −
 have ∀ Σ. mset (map snd (𝔘 Σ Ξ)) = mset ((map snd Σ) ∩ Ξ)
 proof (induct Ξ)
  case Nil
  then show ?case by simp
 next
  case (Cons ξ Ξ)
  {
   fix Σ
   have mset (map snd (𝔘 Σ (ξ # Ξ))) = mset (map snd Σ ∩ ξ # Ξ)
   proof (cases find (λ σ. ξ = snd σ) Σ)
    case None
    hence ξ ∉ set (map snd Σ)
    proof (induct Σ)
     case Nil
     then show ?case by simp
    next
     case (Cons σ Σ)
     have find (λσ. ξ = snd σ) Σ = None
         ξ ≠ snd σ
      using Cons.prems
      by (auto, metis Cons.prems find.simps(2) find-None-iff list.set-intros(1))
     then show ?case using Cons.hyps by simp
    qed
    then show ?thesis using None Cons.hyps by simp
   next
    case (Some σ)
    hence σ ∈ set Σ ξ = snd σ
     by (meson find-Some-predicate find-Some-set-membership)+
    moreover
    from ‹σ ∈ set Σ› have mset Σ = mset (σ # (remove1 σ Σ))
     by simp
    hence mset (map snd Σ) = mset ((snd σ) # (remove1 (snd σ) (map snd
Σ)))
        mset (map snd Σ) = mset (map snd (σ # (remove1 σ Σ)))
     by (simp add: ‹σ ∈ set Σ›, metis map-monotonic subset-mset.eq-iff)
    hence mset (map snd (remove1 σ Σ)) = mset (remove1 (snd σ) (map snd
Σ))
     by simp
```

ultimately show *?thesis* **using** *Some Cons.hyps* **by** *simp*
qed
}
then show *?case* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *witness-list-implication-rule*:
$\vdash$ (*map* (*uncurry* (⊔)) Σ :→ φ) → ⨅ (*map* (λ (χ, ξ). (χ → ξ) → φ) Σ) → φ
proof (*induct* Σ)
case *Nil*
then show *?case* **using** *axiom-k* **by** *simp*
next
case (*Cons* σ Σ)
let *?*χ = *fst* σ
let *?*ξ = *snd* σ
let *?*Σ_A = *map* (*uncurry* (⊔)) Σ
let *?*Σ_B = *map* (λ (χ, ξ). (χ → ξ) → φ) Σ
assume $\vdash$ *?*Σ_A :→ φ → ⨅ *?*Σ_B → φ
moreover have
$\vdash$ (*?*Σ_A :→ φ → ⨅ *?*Σ_B → φ)
→ ((*?*χ ⊔ *?*ξ) → *?*Σ_A :→ φ) → (((*?*χ → *?*ξ) → φ) ⊓ ⨅ *?*Σ_B) → φ
proof −
let *?*φ = (⟨*?*Σ_A :→ φ⟩ → ⟨⨅ *?*Σ_B⟩ → ⟨φ⟩)
→ ((⟨*?*χ⟩ ⊔ ⟨*?*ξ⟩) → ⟨*?*Σ_A :→ φ⟩) → ((⟨*?*χ⟩ → ⟨*?*ξ⟩) → ⟨φ⟩) ⊓ ⟨⨅ *?*Σ_B⟩) → ⟨φ⟩)
have ∀ 𝔐. 𝔐 ⊨$_{prop}$ *?*φ **by** *fastforce*
hence $\vdash$ ⦇ *?*φ ⦈ **using** *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
ultimately have $\vdash$ ((*?*χ ⊔ *?*ξ) → *?*Σ_A :→ φ) → (((*?*χ → *?*ξ) → φ) ⊓ ⨅ *?*Σ_B) → φ
using *modus-ponens* **by** *blast*
moreover
have (λ σ. (*fst* σ → *snd* σ) → φ) = (λ (χ, ξ). (χ → ξ) → φ)
uncurry (⊔) = (λ σ. *fst* σ ⊔ *snd* σ)
by *fastforce*+
hence (λ (χ, ξ). (χ → ξ) → φ) σ = (*?*χ → *?*ξ) → φ
uncurry (⊔) σ = *?*χ ⊔ *?*ξ
by *metis*+
ultimately show *?case* **by** *simp*
qed

lemma (**in** *classical-logic*) *witness-relative-MaxSAT-increase*:
assumes ¬ $\vdash$ φ
and *mset* (*map snd* Σ) ⊆# *mset* Γ
and *map* (*uncurry* (⊔)) Σ :⊢ φ
shows (| Γ |$_\varphi$) < (| *map* (*uncurry* (→)) Σ @ Γ ⊖ *map snd* Σ |$_\varphi$)

proof −
from ‹$\neg \vdash \varphi$› **obtain** Ξ **where** Ξ: $\Xi \in \mathcal{M}\ \Gamma\ \varphi$
using *relative-maximals-existence* **by** *blast*
let $?\Sigma' = \Sigma \ominus \mathfrak{U}\ \Sigma\ \Xi$
let $?\Sigma\Xi' = map\ (uncurry\ (\sqcup))\ (\mathfrak{U}\ \Sigma\ \Xi)\ @\ map\ (uncurry\ (\rightarrow))\ (\mathfrak{U}\ \Sigma\ \Xi)$
have $mset\ \Sigma = mset\ (\mathfrak{U}\ \Sigma\ \Xi\ @\ ?\Sigma')$ **by** (*simp add*: *MaxSAT-witness-left-msub*)
hence $set\ (map\ (uncurry\ (\sqcup))\ \Sigma) = set\ (map\ (uncurry\ (\sqcup))\ ((\mathfrak{U}\ \Sigma\ \Xi)\ @\ ?\Sigma'))$
by (*metis mset-map mset-eq-setD*)
hence $map\ (uncurry\ (\sqcup))\ ((\mathfrak{U}\ \Sigma\ \Xi)\ @\ ?\Sigma') :\vdash \varphi$
using *list-deduction-monotonic assms*(*3*)
by *blast*
hence $map\ (uncurry\ (\sqcup))\ (\mathfrak{U}\ \Sigma\ \Xi)\ @\ map\ (uncurry\ (\sqcup))\ ?\Sigma' :\vdash \varphi$ **by** *simp*
moreover
{
fix $\Phi\ \Psi$
have $((\Phi\ @\ \Psi) :\rightarrow \varphi) = (\Phi :\rightarrow (\Psi :\rightarrow \varphi))$
by (*induct* Φ, *simp*+)
hence $(\Phi\ @\ \Psi) :\vdash \varphi = \Phi :\vdash (\Psi :\rightarrow \varphi)$
unfolding *list-deduction-def*
by (*induct* Φ, *simp*+)
}
ultimately have $map\ (uncurry\ (\sqcup))\ (\mathfrak{U}\ \Sigma\ \Xi) :\vdash map\ (uncurry\ (\sqcup))\ ?\Sigma' :\rightarrow \varphi$
by *simp*
moreover have $set\ (map\ (uncurry\ (\sqcup))\ (\mathfrak{U}\ \Sigma\ \Xi)) \subseteq set\ ?\Sigma\Xi'$
by *simp*
ultimately have $?\Sigma\Xi' :\vdash map\ (uncurry\ (\sqcup))\ ?\Sigma' :\rightarrow \varphi$
using *list-deduction-monotonic* **by** *blast*
hence $?\Sigma\Xi' :\vdash \bigsqcap\ (map\ (\lambda\ (\chi, \gamma).\ (\chi \rightarrow \gamma) \rightarrow \varphi)\ ?\Sigma') \rightarrow \varphi$
using *list-deduction-modus-ponens*
list-deduction-weaken
witness-list-implication-rule
by *blast*
hence $?\Sigma\Xi'\ \$\vdash [\bigsqcap\ (map\ (\lambda\ (\chi, \gamma).\ (\chi \rightarrow \gamma) \rightarrow \varphi)\ ?\Sigma') \rightarrow \varphi]$
using *measure-deduction-one-collapse* **by** *metis*
hence
$?\Sigma\Xi'\ @\ (map\ snd\ (\mathfrak{U}\ \Sigma\ \Xi)) \ominus (map\ snd\ (\mathfrak{U}\ \Sigma\ \Xi))$
$\$\vdash [\bigsqcap\ (map\ (\lambda\ (\chi, \gamma).\ (\chi \rightarrow \gamma) \rightarrow \varphi)\ ?\Sigma') \rightarrow \varphi]$
by *simp*
hence $map\ snd\ (\mathfrak{U}\ \Sigma\ \Xi)\ \$\vdash [\bigsqcap\ (map\ (\lambda\ (\chi, \gamma).\ (\chi \rightarrow \gamma) \rightarrow \varphi)\ ?\Sigma') \rightarrow \varphi]$
using *measure-witness-left-split* [**where** Γ=$map\ snd\ (\mathfrak{U}\ \Sigma\ \Xi)$
and Σ=$\mathfrak{U}\ \Sigma\ \Xi$]
by *fastforce*
hence $map\ snd\ (\mathfrak{U}\ \Sigma\ \Xi)\ \$\vdash [\bigsqcap\ (map\ (\lambda\ (\chi, \gamma).\ (\chi \rightarrow \gamma) \rightarrow \varphi)\ ?\Sigma') \rightarrow \varphi]$
using *MaxSAT-witness-right-projection* **by** *auto*
hence $map\ snd\ (\mathfrak{U}\ \Sigma\ \Xi) :\vdash \bigsqcap\ (map\ (\lambda\ (\chi, \gamma).\ (\chi \rightarrow \gamma) \rightarrow \varphi)\ ?\Sigma') \rightarrow \varphi$
using *measure-deduction-one-collapse* **by** *blast*
hence $\star$:
$map\ snd\ (\mathfrak{U}\ \Sigma\ \Xi)\ @\ \Xi \ominus (map\ snd\ \Sigma) :\vdash \bigsqcap\ (map\ (\lambda\ (\chi, \gamma).\ (\chi \rightarrow \gamma) \rightarrow \varphi)\ ?\Sigma') \rightarrow \varphi$

(**is** $?\Xi_0 \vdash$ -)
using *list-deduction-monotonic*
by (*metis* (*no-types*, *lifting*) *append-Nil2*
measure-cancel
measure-deduction.simps(*1*)
measure-list-deduction-antitonic)
have *mset* Ξ = *mset* ($\Xi \ominus$ (*map snd* Σ)) + *mset* ($\Xi \cap$ (*map snd* Σ))
using *list-diff-intersect-comp* **by** *blast*
hence *mset* Ξ = *mset* ((*map snd* Σ) $\cap$ Ξ) + *mset* ($\Xi \ominus$ (*map snd* Σ))
by (*metis subset-mset.inf-commute list-intersect-mset-homomorphism union-commute*)
hence *mset* Ξ = *mset* (*map snd* ($\mathfrak{U}$ Σ Ξ)) + *mset* ($\Xi \ominus$ (*map snd* Σ))
using *MaxSAT-witness-right-projection* **by** *simp*
hence *mset* Ξ = *mset* $?\Xi_0$
by *simp*
hence *set* Ξ = *set* $?\Xi_0$
by (*metis mset-eq-setD*)
have $\neg$ $?\Xi_0 \vdash \bigsqcap$ (*map* (λ (χ, γ). ($\chi \to \gamma$) $\to$ φ) $?\Sigma'$)
proof (*rule notI*)
assume $?\Xi_0 \vdash \bigsqcap$ (*map* (λ (χ, γ). ($\chi \to \gamma$) $\to$ φ) $?\Sigma'$)
hence $?\Xi_0 \vdash \varphi$
using $\star$ *list-deduction-modus-ponens* **by** *blast*
hence $\Xi \vdash \varphi$
using *list-deduction-monotonic* ‹*set* Ξ = *set* $?\Xi_0$› **by** *blast*
thus *False*
using Ξ *relative-maximals-def* **by** *blast*
qed
moreover
have *mset* (*map snd* ($\mathfrak{U}$ Σ Ξ)) $\subseteq\#$ *mset* $?\Xi_0$
mset (*map* (*uncurry* ($\to$)) ($\mathfrak{U}$ Σ Ξ) @ $?\Xi_0 \ominus$ *map snd* ($\mathfrak{U}$ Σ Ξ))
= *mset* (*map* (*uncurry* ($\to$)) ($\mathfrak{U}$ Σ Ξ) @ $\Xi \ominus$ (*map snd* Σ))
(**is** - = *mset* $?\Xi_1$)
by *auto*
hence $?\Xi_1 \preceq ?\Xi_0$
by (*metis add.commute*
witness-stronger-theory
add-diff-cancel-right'
list-subtract.simps(*1*)
list-subtract-mset-homomorphism
list-diff-intersect-comp
list-intersect-right-project
msub-stronger-theory-intro
stronger-theory-combine
stronger-theory-empty-list-intro
self-append-conv)
ultimately have
$\neg$ $?\Xi_1 \vdash \bigsqcap$ (*map* (λ (χ, γ). ($\chi \to \gamma$) $\to$ φ) $?\Sigma'$)
using *stronger-theory-deduction-monotonic* **by** *blast*
from *this* **obtain** χ γ **where**
(χ,γ) $\in$ *set* $?\Sigma'$

```
    ¬ (χ → γ) # ?Ξ₁ ⊢ φ
    using list-deduction-theorem
    by fastforce
  have mset (χ → γ # ?Ξ₁) ⊆# mset (map (uncurry (→)) Σ @ Γ ⊖ map snd Σ)
  proof −
    let ?A = map (uncurry (→)) Σ
    let ?B = map (uncurry (→)) (𝔘 Σ Ξ)
    have (χ,γ) ∈ (set Σ − set (𝔘 Σ Ξ))
    proof −
      from ‹(χ,γ) ∈ set ?Σ'› have γ ∈# mset (map snd (Σ ⊖ 𝔘 Σ Ξ))
        by (metis set-mset-mset image-eqI set-map snd-conv)
      hence γ ∈# mset (map snd Σ ⊖ map snd (𝔘 Σ Ξ))
        by (metis MaxSAT-witness-left-msub map-list-subtract-mset-equivalence)
      hence γ ∈# mset (map snd Σ ⊖ (map snd Σ ∩ Ξ))
      by (metis MaxSAT-witness-right-projection list-subtract-mset-homomorphism)
      hence γ ∈# mset (map snd Σ ⊖ Ξ)
        by (metis add-diff-cancel-right'
                  list-subtract-mset-homomorphism
                  list-diff-intersect-comp)
      moreover from assms(2) have mset (map snd Σ ⊖ Ξ) ⊆# mset (Γ ⊖ Ξ)
        by (simp, metis list-subtract-monotonic list-subtract-mset-homomorphism
mset-map)
      ultimately have γ ∈# mset (Γ ⊖ Ξ)
        by (simp add: mset-subset-eqD)
      hence γ ∈ set (Γ ⊖ Ξ)
        using set-mset-mset by fastforce
      hence γ ∈ set Γ − set Ξ
        using Ξ by simp
      hence γ ∉ set Ξ
        by blast
      hence ∀ Σ. (χ,γ) ∉ set (𝔘 Σ Ξ)
      proof (induct Ξ)
        case Nil
        then show ?case by simp
      next
        case (Cons ξ Ξ)
        {
          fix Σ
          have (χ, γ) ∉ set (𝔘 Σ (ξ # Ξ))
          proof (cases find (λσ. ξ = snd σ) Σ)
            case None
            then show ?thesis using Cons by simp
          next
            case (Some σ)
            moreover from this have snd σ = ξ
              using find-Some-predicate by fastforce
            with Cons.prems have σ ≠ (χ,γ) by fastforce
            ultimately show ?thesis using Cons by simp
          qed
```

```
    }
    then show ?case by blast
  qed
  moreover from ‹(χ,γ) ∈ set ?Σ'› have (χ,γ) ∈ set Σ
    by (meson list-subtract-set-trivial-upper-bound subsetCE)
  ultimately show ?thesis by fastforce
 qed
 with ‹(χ, γ) ∈ set ?Σ'› have mset ((χ,γ) # 𝔄 Σ Ξ) ⊆# mset Σ
  by (meson MaxSAT-witness-left-msub msub-list-subtract-elem-cons-msub)
 hence mset (χ → γ # ?B) ⊆# mset (map (uncurry (→)) Σ)
  by (metis (no-types, lifting)
       ‹(χ, γ) ∈ set ?Σ'›
       MaxSAT-witness-left-msub
       map-list-subtract-mset-equivalence
       map-monotonic
       mset-eq-setD msub-list-subtract-elem-cons-msub
       pair-imageI
       set-map
       uncurry-def)
 moreover
 have mset Ξ ⊆# mset Γ
  using Ξ relative-maximals-def
  by blast
 hence mset (Ξ ⊖ (map snd Σ)) ⊆# mset (Γ ⊖ (map snd Σ))
  using list-subtract-monotonic by blast
 ultimately show ?thesis
  using subset-mset.add-mono by fastforce
qed
moreover have length ?Ξ₁ = length ?Ξ₀
 by simp
hence length ?Ξ₁ = length Ξ
 using ‹mset Ξ = mset ?Ξ₀› mset-eq-length
 by metis
hence length ((χ → γ) # ?Ξ₁) = length Ξ + 1
 by simp
hence length ((χ → γ) # ?Ξ₁) = (| Γ |_φ) + 1
 using Ξ
 by (simp add: relative-MaxSAT-intro)
moreover from ‹¬ ⊢ φ› obtain Ω where Ω: Ω ∈ 𝓜 (map (uncurry (→)) Σ @
Γ ⊖ map snd Σ) φ
 using relative-maximals-existence by blast
ultimately have length Ω ≥ (| Γ |_φ) + 1
 using relative-maximals-def
 by (metis (no-types, lifting) ‹¬ χ → γ # ?Ξ₁ :⊢ φ› mem-Collect-eq)
thus ?thesis
 using Ω relative-MaxSAT-intro by auto
qed

lemma (in classical-logic) relative-maximals-counting-deduction-lower-bound:
```

```
  assumes ¬ ⊢ φ
    shows (Γ #⊢ n φ) = (n ≤ ‖ Γ ‖φ)
proof −
  have ∀ Γ. (Γ #⊢ n φ) = (n ≤ ‖ Γ ‖φ)
  proof (induct n)
    case 0
    then show ?case by simp
  next
    case (Suc n)
    {
      fix Γ
      assume Γ #⊢ (Suc n) φ
      from this obtain Σ where Σ:
        mset (map snd Σ) ⊆# mset Γ
        map (uncurry (⊔)) Σ :⊢ φ
        map (uncurry (→)) Σ @ Γ ⊖ (map snd Σ) #⊢ n φ
        by fastforce
      let ?Γ′ = map (uncurry (→)) Σ @ Γ ⊖ (map snd Σ)
      have length Γ = length ?Γ′
        using Σ(1) list-subtract-msub-eq by fastforce
      hence (‖ Γ ‖φ) > (‖ ?Γ′ ‖φ)
        by (metis Σ(1) Σ(2) ‹¬ ⊢ φ›
                  witness-relative-MaxSAT-increase
                  length-MaxSAT-decomposition
                  add-less-cancel-right
                  nat-add-left-cancel-less)
      with Σ(3) Suc.hyps have Suc n ≤ ‖ Γ ‖φ
        by auto
    }
    moreover
    {
      fix Γ
      assume Suc n ≤ ‖ Γ ‖φ
      from this obtain Σ where Σ:
        mset (map snd Σ) ⊆# mset Γ
        map (uncurry (⊔)) Σ :⊢ φ
        1 + (‖ map (uncurry (→)) Σ @ Γ ⊖ map snd Σ ‖φ) = ‖ Γ ‖φ
        (is 1 + (‖ ?Γ′ ‖φ) = ‖ Γ ‖φ)
        by (metis Suc-le-D assms relative-maximals-optimal-witness zero-less-Suc)
      have n ≤ ‖ ?Γ′ ‖φ
        using Σ(3) ‹Suc n ≤ ‖ Γ ‖φ› by linarith
      hence ?Γ′ #⊢ n φ using Suc by blast
      hence Γ #⊢ (Suc n) φ using Σ(1) Σ(2) by fastforce
    }
    ultimately show ?case by metis
  qed
  thus ?thesis by auto
qed
```

As a brief aside, we may observe that φ is a tautology if and only if count-

ing deduction can prove it for any given number of times. This follows immediately from $\neg \vdash \varphi \Longrightarrow \Gamma \mathrel{\#\vdash} n\ \varphi = (n \leq \| \Gamma \|_\varphi)$.

lemma (**in** *classical-logic*) *counting-deduction-tautology-equiv*:
$(\forall\ n.\ \Gamma \mathrel{\#\vdash} n\ \varphi) = \vdash \varphi$
proof (*cases* $\vdash \varphi$)
case *True*
then show *?thesis*
by (*simp add*: *counting-deduction-tautology-weaken*)
next
case *False*
have $\neg\ \Gamma \mathrel{\#\vdash} (1 + \mathit{length}\ \Gamma)\ \varphi$
proof (*rule notI*)
assume $\Gamma \mathrel{\#\vdash} (1 + \mathit{length}\ \Gamma)\ \varphi$
hence $1 + \mathit{length}\ \Gamma \leq \| \Gamma \|_\varphi$
using ‹$\neg \vdash \varphi$› *relative-maximals-counting-deduction-lower-bound* **by** *blast*
hence $1 + \mathit{length}\ \Gamma \leq \mathit{length}\ \Gamma$
using *complement-relative-MaxSAT-def* **by** *fastforce*
thus *False* **by** *linarith*
qed
then show *?thesis*
using ‹$\neg \vdash \varphi$› **by** *blast*
qed

theorem (**in** *classical-logic*) *relative-maximals-max-counting-deduction*:
$\Gamma \mathrel{\#\vdash} n\ \varphi = (\forall\ \Phi \in \mathcal{M}\ \Gamma\ \varphi.\ n \leq \mathit{length}\ (\Gamma \ominus \Phi))$
proof (*cases* $\vdash \varphi$)
case *True*
from ‹$\vdash \varphi$› **have** $\Gamma \mathrel{\#\vdash} n\ \varphi$
using *counting-deduction-tautology-weaken*
by *blast*
moreover from ‹$\vdash \varphi$› **have** $\mathcal{M}\ \Gamma\ \varphi = \{\}$
using *relative-maximals-existence* **by** *auto*
hence $\forall\ \Phi \in \mathcal{M}\ \Gamma\ \varphi.\ n \leq \mathit{length}\ (\Gamma \ominus \Phi)$ **by** *blast*
ultimately show *?thesis* **by** *meson*
next
case *False*
from ‹$\neg \vdash \varphi$› **have** $(\Gamma \mathrel{\#\vdash} n\ \varphi) = (n \leq \| \Gamma \|_\varphi)$
by (*simp add*: *relative-maximals-counting-deduction-lower-bound*)
moreover have $(n \leq \| \Gamma \|_\varphi) = (\forall\ \Phi \in \mathcal{M}\ \Gamma\ \varphi.\ n \leq \mathit{length}\ (\Gamma \ominus \Phi))$
proof (*rule iffI*)
assume $n \leq \| \Gamma \|_\varphi$
{
fix Φ
assume $\Phi \in \mathcal{M}\ \Gamma\ \varphi$
hence $n \leq \mathit{length}\ (\Gamma \ominus \Phi)$
using ‹$n \leq \| \Gamma \|_\varphi$› *complement-relative-MaxSAT-intro* **by** *auto*
}
thus $\forall\, \Phi \in \mathcal{M}\ \Gamma\ \varphi.\ n \leq \mathit{length}\ (\Gamma \ominus \Phi)$ **by** *blast*
next

```
    assume ∀ Φ ∈ 𝓜 Γ φ. n ≤ length (Γ ⊖ Φ)
    with ‹¬ ⊢ φ› obtain Φ where
      Φ ∈ 𝓜 Γ φ
      n ≤ length (Γ ⊖ Φ)
      using relative-maximals-existence
      by blast
    thus n ≤ ‖ Γ ‖φ
      by (simp add: complement-relative-MaxSAT-intro)
  qed
  ultimately show ?thesis by metis
qed

lemma (in consistent-classical-logic) counting-deduction-to-maxsat:
  (Γ #⊢ n ⊥) = (MaxSAT Γ + n ≤ length Γ)
  by (metis
        add.commute
        consistency
        length-MaxSAT-decomposition
        relative-maximals-counting-deduction-lower-bound
        nat-add-left-cancel-le)
```

Chapter 4

Inequality Completeness For Probability Logic

4.1 Limited Counting Deduction Completeness

The reduction of counting deduction to MaxSAT allows us to first prove completeness for counting deduction, as maximal consistent sublists allow us to recover maximally consistent sets, which give rise to Dirac measures.

The completeness result first presented here, where all of the propositions on the left hand side are the same, will be extended later.

lemma (**in** *probability-logic*) *list-probability-upper-bound*:
  $(\sum \gamma \leftarrow \Gamma.\ \mathcal{P}\ \gamma) \leq$ *real* (*length* Γ)
proof (*induct* Γ)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons* γ Γ)
  moreover have $\mathcal{P}\ \gamma \leq 1$ **using** *unity-upper-bound* **by** *blast*
  ultimately have $\mathcal{P}\ \gamma + (\sum \gamma \leftarrow \Gamma.\ \mathcal{P}\ \gamma) \leq 1 +$ *real* (*length* Γ) **by** *linarith*
  then show *?case* **by** *simp*
qed

theorem (**in** *classical-logic*) *dirac-limited-counting-deduction-completeness*:
  $(\forall\ \mathcal{P} \in$ *dirac-measures*. *real* $n * \mathcal{P}\ \varphi \leq (\sum \gamma \leftarrow \Gamma.\ \mathcal{P}\ \gamma)) = {\sim}\ \Gamma\ \#\vdash\ n\ (\sim \varphi)$
proof −
  {
    fix $\mathcal{P}$ :: $'a \Rightarrow$ *real*
    assume $\mathcal{P} \in$ *dirac-measures*
    from *this* **interpret** *probability-logic* $(\lambda\ \varphi.\ \vdash \varphi)\ (\rightarrow)\ \bot\ \mathcal{P}$
      unfolding *dirac-measures-def*
      by *auto*
    assume ${\sim}\ \Gamma\ \#\vdash\ n\ (\sim \varphi)$
    moreover have *replicate* $n\ (\sim \varphi) = {\sim}\ (\mathit{replicate}\ n\ \varphi)$

```
    by (induct n, auto)
  ultimately have ∼ Γ $⊢ ∼ (replicate n φ)
    using counting-deduction-to-measure-deduction by metis
  hence (∑ φ←(replicate n φ). P φ) ≤ (∑ γ←Γ. P γ)
    using measure-deduction-soundness
    by blast
  moreover have (∑ φ←(replicate n φ). P φ) = real n * P φ
    by (induct n, simp, simp add: semiring-normalization-rules(3))
  ultimately have real n * P φ ≤ (∑ γ←Γ. P γ)
    by simp
}
moreover
{
  assume ¬ ∼ Γ #⊢ n (∼ φ)
  have ∃ P ∈ dirac-measures. real n * P φ > (∑ γ←Γ. P γ)
  proof −
    have ∃ Φ. Φ ∈ M (∼ Γ) (∼ φ)
      using
        ‹¬ ∼ Γ #⊢ n (∼ φ)›
        relative-maximals-existence
        counting-deduction-tautology-weaken
      by blast
    from this obtain Φ where Φ:
      (∼ Φ) ∈ M (∼ Γ) (∼ φ)
      mset Φ ⊆# mset Γ
      unfolding map-negation-def
      by (metis
            (mono-tags, lifting)
            relative-maximals-def
            mem-Collect-eq
            mset-sub-map-list-exists)
    hence ¬ ⊢ φ → ⨆ Φ
      using
        biconditional-weaken
        list-deduction-def
        map-negation-list-implication
        set-deduction-base-theory
        relative-maximals-def
      by blast
    from this obtain Ω where Ω: MCS Ω φ ∈ Ω ⨆ Φ ∉ Ω
      by (meson
            insert-subset
            formula-consistent-def
            formula-maximal-consistency
            formula-maximally-consistent-extension
            formula-maximally-consistent-set-def-def
            set-deduction-base-theory
            set-deduction-reflection
            set-deduction-theorem)
```

let *?P* = λ *χ*. *if* *χ*∈Ω *then* (*1* :: *real*) *else* *0*
from Ω **have** *?P* ∈ *dirac-measures*
using *MCS-dirac-measure* **by** *blast*
moreover
from *this* **interpret** *probability-logic* (λ *φ*. ⊢ *φ*) (→) ⊥ *?P*
unfolding *dirac-measures-def*
by *auto*
have ∀ *φ* ∈ *set* Φ. *?P* *φ* = *0*
using Φ(*1*) Ω(*1*) Ω(*3*) *arbitrary-disjunction-exclusion-MCS* **by** *auto*
with Φ(*2*) **have** (∑ γ←Γ. *?P* γ) = (∑ γ←(Γ ⊖ Φ). *?P* γ)
proof (*induct* Φ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* *φ* Φ)
then show *?case*
proof −
obtain *ω* :: *'a* **where**
ω: ¬ *mset* Φ ⊆# *mset* Γ
∨ *ω* ∈ *set* Φ ∧ *ω* ∈ Ω
∨ (∑ γ←Γ. *?P* γ) = (∑ γ←Γ ⊖ Φ. *?P* γ)
using *Cons.hyps* **by** *fastforce*
have *A*:
∀(*f* :: *'a* ⇒ *real*) (Γ ::*'a list*) Φ.
¬ *mset* Φ ⊆# *mset* Γ
∨ *sum-list* ((∑ φ←Φ. *f* *φ*) # *map f* (Γ ⊖ Φ)) = (∑ γ←Γ. *f* γ)
using *listSubstract-multisubset-list-summation* **by** *auto*
have *B*: ∀ *rs*. *sum-list* ((*0*::*real*) # *rs*) = *sum-list* *rs*
by *auto*
have *C*: ∀ *r* *rs*. (*0*::*real*) = *r* ∨ *sum-list* (*r* # *rs*) ≠ *sum-list* *rs*
by *simp*
have *D*: ∀ *f*. *sum-list* (*sum-list* (*map f* (*φ* # Φ)) # *map f* (Γ ⊖ (*φ* # Φ)))
= (*sum-list* (*map f* Γ)::*real*)
using *A* *Cons.prems*(*1*) **by** *blast*
have *E*: *mset* Φ ⊆# *mset* Γ
using *Cons.prems*(*1*) *subset-mset.dual-order.trans* **by** *force*
then have *F*: ∀ *f*. (*0*::*real*) = *sum-list* (*map f* Φ)
∨ *sum-list* (*map f* Γ) ≠ *sum-list* (*map f* (Γ ⊖ Φ))
using *C* *A* **by** (*metis* (*no-types*))
then have *G*: (∑ *φ'*←(*φ* # Φ). *?P* *φ'*) = *0* ∨ *ω* ∈ Ω
using *E* *ω* *Cons.prems*(*2*) **by** *auto*
have *H*: ∀ Γ *r*::*real*. *r* = (∑ γ←Γ. *?P* γ)
∨ *ω* ∈ *set* Φ
∨ *r* ≠ (∑ γ←(*φ* # Γ). *?P* γ)
using *Cons.prems*(*2*) **by** *auto*
have (*1*::*real*) ≠ *0* **by** *linarith*
moreover
{ **assume** *ω* ∉ *set* Φ
then have *ω* ∉ Ω ∨ (∑ γ←Γ. *?P* γ) = (∑ γ←Γ ⊖ (*φ* # Φ). *?P* γ)

```
        using H F E D B ω by (metis (no-types) sum-list.Cons) }
      ultimately have ?thesis
        using G D B by (metis Cons.prems(2) list.set-intros(2))
      then show ?thesis
        by linarith
    qed
  qed
  hence (∑γ←Γ. ?P γ) ≤ real (length (Γ ⊖ Φ))
    using list-probability-upper-bound
    by auto
      moreover
  have length (∼ Γ ⊖ ∼ Φ) < n
    by (metis not-le Φ(1) ‹¬ (∼ Γ) #⊢ n (∼ φ)›
          relative-maximals-max-counting-deduction
          maximals-list-subtract-length-equiv)
  hence real (length (∼ Γ ⊖ ∼ Φ)) < real n
    by simp
  with Ω(2) have real (length (∼ Γ ⊖ ∼ Φ)) < real n * ?P φ
    by simp
  moreover
  have (∼ (Γ ⊖ Φ)) ⇌ (∼ Γ ⊖ ∼ Φ)
    unfolding map-negation-def
    by (metis Φ(2) map-list-subtract-mset-equivalence)
  with perm-length have length (Γ ⊖ Φ) = length (∼ Γ ⊖ ∼ Φ)
    by (metis length-map local.map-negation-def)
  hence real (length (Γ ⊖ Φ)) = real (length (∼ Γ ⊖ ∼ Φ))
    by simp
  ultimately show ?thesis
    by force
 qed
}
ultimately show ?thesis by fastforce
qed
```

4.2 Measure Deduction Completeness

Since measure deduction may be reduced to counting deduction, we have measure deduction is complete.

```
lemma (in classical-logic) dirac-measure-deduction-completeness:
  (∀ P ∈ dirac-measures. (∑φ←Φ. P φ) ≤ (∑γ←Γ. P γ)) = ∼ Γ $⊢ ∼ Φ
proof −
  {
    fix P :: 'a ⇒ real
    assume P ∈ dirac-measures
    from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ P
      unfolding dirac-measures-def
      by auto
    assume ∼ Γ $⊢ ∼ Φ
```

```
    hence (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←Γ. 𝒫 γ)
      using measure-deduction-soundness
      by blast
  }
  moreover
  {
    assume ¬ ∼ Γ $⊢ ∼ Φ
    have ∃ 𝒫 ∈ dirac-measures. (∑ φ←Φ. 𝒫 φ) > (∑ γ←Γ. 𝒫 γ)
    proof −
      from ‹¬ ∼ Γ $⊢ ∼ Φ› have ¬ ∼ (∼ Φ) @ ∼ Γ #⊢ (length (∼ Φ)) ⊥
        using measure-deduction-to-counting-deduction by blast
      moreover
      have ∼ (∼ Φ) @ ∼ Γ #⊢ (length (∼ Φ)) ⊥ = ∼ (∼ Φ) @ ∼ Γ #⊢ (length
Φ) ⊥
        by (induct Φ, auto)
      moreover have ⊢ ∼ ⊤ → ⊥
        by (simp add: negation-def)
      ultimately have ¬ ∼ (∼ Φ @ Γ) #⊢ (length Φ) (∼ ⊤)
        using counting-deduction-implication by fastforce
      from this obtain 𝒫 where 𝒫:
        𝒫 ∈ dirac-measures
        real (length Φ) * 𝒫 ⊤ > (∑ γ← (∼ Φ @ Γ). 𝒫 γ)
        using dirac-limited-counting-deduction-completeness
        by fastforce
      from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ 𝒫
        unfolding dirac-measures-def
        by auto
      from 𝒫(2) have real (length Φ) > (∑ γ← ∼ Φ. 𝒫 γ) + (∑ γ← Γ. 𝒫 γ)
        by (simp add: probability-unity)
      moreover have (∑ γ← ∼ Φ. 𝒫 γ) = real (length Φ) − (∑ γ← Φ. 𝒫 γ)
        using complementation
        by (induct Φ, auto)
      ultimately show ?thesis
        using 𝒫(1) by auto
    qed
  }
  ultimately show ?thesis by fastforce
qed

theorem (in classical-logic) measure-deduction-completeness:
  (∀ 𝒫 ∈ probabilities. (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←Γ. 𝒫 γ)) = ∼ Γ $⊢ ∼ Φ
proof −
  {
    fix 𝒫 :: 'a ⇒ real
    assume 𝒫 ∈ probabilities
    from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ 𝒫
      unfolding probabilities-def
      by auto
    assume ∼ Γ $⊢ ∼ Φ
```

hence $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
using *measure-deduction-soundness*
by *blast*
}
thus *?thesis*
using *dirac-measures-subset dirac-measure-deduction-completeness*
by *fastforce*
qed

4.3 Counting Deduction Completeness

Leveraging our measure deduction completeness result, we may extend our limited counting deduction completeness theorem to full completness.

lemma (**in** *classical-logic*) *measure-left-commute*:
$(\Phi\ @\ \Psi)\ \$\vdash\ \Xi = (\Psi\ @\ \Phi)\ \$\vdash\ \Xi$
proof −
have $(\Phi\ @\ \Psi) \preceq (\Psi\ @\ \Phi)\ (\Psi\ @\ \Phi) \preceq (\Phi\ @\ \Psi)$
using *stronger-theory-reflexive stronger-theory-right-permutation perm-append-swap*
by *blast+*
thus *?thesis*
using *measure-stronger-theory-left-monotonic*
by *blast*
qed

lemma (**in** *classical-logic*) *stronger-theory-double-negation-right*:
$\Phi \preceq \sim (\sim \Phi)$
by (*induct* Φ, *simp*, *simp add*: *double-negation negation-def stronger-theory-left-right-cons*)

lemma (**in** *classical-logic*) *stronger-theory-double-negation-left*:
$\sim (\sim \Phi) \preceq \Phi$
by (*induct* Φ,
simp,
simp add: *double-negation-converse negation-def stronger-theory-left-right-cons*)

lemma (**in** *classical-logic*) *counting-deduction-completeness*:
$(\forall\ \mathcal{P} \in$ *dirac-measures*. $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)) = (\sim \Gamma\ @\ \Phi)\ \#\vdash$
$(\mathit{length}\ \Phi)\ \bot$
proof −
have $(\forall\ \mathcal{P} \in$ *dirac-measures*. $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
$= \sim (\sim \Phi)\ @\ \sim \Gamma\ \#\vdash\ (\mathit{length}\ (\sim \Phi))\ \bot$
using *dirac-measure-deduction-completeness measure-deduction-to-counting-deduction*
by *blast*
also have ... $= \sim (\sim \Phi)\ @\ \sim \Gamma\ \#\vdash\ (\mathit{length}\ \Phi)\ \bot$ **by** (*induct* Φ, *auto*)
also have ... $= \sim \Gamma\ @\ \sim (\sim \Phi)\ \#\vdash\ (\mathit{length}\ \Phi)\ \bot$
by (*simp add*: *measure-left-commute counting-deduction-to-measure-deduction*)
also have ... $= \sim \Gamma\ @\ \Phi\ \#\vdash\ (\mathit{length}\ \Phi)\ \bot$
by (*meson measure-cancel*
stronger-theory-to-measure-deduction

measure-transitive
counting-deduction-to-measure-deduction
stronger-theory-double-negation-left
stronger-theory-double-negation-right)
finally show *?thesis* **by** *blast*
qed

4.4 Collapse Theorem For Probability Logic

We now turn to proving the collapse theorem for probability logic. This states that any inequality holds for all finitely additive probability measures if and only if it holds for all Dirac measures.

theorem (**in** *classical-logic*) *weakly-additive-completeness-collapse*:
$(\forall\ \mathcal{P} \in \mathit{probabilities}.\ (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
$= (\forall\ \mathcal{P} \in \mathit{dirac\text{-}measures}.\ (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
by (*simp add*: *dirac-measure-deduction-completeness*
measure-deduction-completeness)

The collapse theorem may be strengthened to include an arbitrary constant term c. This will be key to characterizing MaxSAT completeness in §4.5.

lemma (**in** *classical-logic*) *nat-dirac-probability*:
$\forall\ \mathcal{P} \in \mathit{dirac\text{-}measures}.\ \exists\, n :: \mathit{nat}.\ \mathit{real}\ n = (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi)$
proof (*induct* Φ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* φ Φ)
{
fix $\mathcal{P} :: 'a \Rightarrow \mathit{real}$
assume $\mathcal{P} \in \mathit{dirac\text{-}measures}$
from *Cons this* **obtain** n **where** $\mathit{real}\ n = (\sum \varphi'{\leftarrow}\Phi.\ \mathcal{P}\ \varphi')$ **by** *fastforce*
hence $\star$: $(\sum \varphi'{\leftarrow}\Phi.\ \mathcal{P}\ \varphi') = \mathit{real}\ n$ **by** *simp*
have $\exists\ n.\ \mathit{real}\ n = (\sum \varphi'{\leftarrow}(\varphi\ \#\ \Phi).\ \mathcal{P}\ \varphi')$
proof (*cases* $\mathcal{P}\ \varphi = 1$)
case *True*
then show *?thesis*
by (*simp add*: $\star$, *metis of-nat-Suc*)
next
case *False*
hence $\mathcal{P}\ \varphi = 0$ **using** ‹$\mathcal{P} \in \mathit{dirac\text{-}measures}$› *dirac-measures-def* **by** *auto*
then show *?thesis* **using** $\star$
by *simp*
qed
}
thus *?case* **by** *blast*
qed

lemma (**in** *classical-logic*) *dirac-ceiling*:
 $\forall\ \mathcal{P} \in$ *dirac-measures*.
  $((\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
   $= ((\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + \lceil c \rceil \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
proof −
 {
  fix $\mathcal{P}$
  assume $\mathcal{P} \in$ *dirac-measures*
  have $((\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
      $= ((\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + \lceil c \rceil \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
  proof (*rule iffI*)
   assume *assm*: $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
   show $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + \lceil c \rceil \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
   proof (*rule ccontr*)
    assume $\neg\ (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + \lceil c \rceil \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
    moreover
    obtain $x :: int$
     and $y :: int$
     and $z :: int$
     where *xyz*: $x = (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi)$
           $y = \lceil c \rceil$
           $z = (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
     using *nat-dirac-probability*
     by (*metis* ‹$\mathcal{P} \in$ *dirac-measures*› *of-int-of-nat-eq*)
    ultimately have $x + y - 1 \geq z$ **by** *linarith*
    hence $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c > (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$ **using** *xyz* **by** *linarith*
    thus *False* **using** *assm* **by** *simp*
   qed
  next
   assume $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + \lceil c \rceil \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
   thus $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
    by *linarith*
  qed
 }
 thus *?thesis* **by** *blast*
qed

lemma (**in** *probability-logic*) *probability-replicate-verum*:
 fixes $n :: nat$
 shows $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + n = (\sum \varphi{\leftarrow}(replicate\ n\ \top)\ @\ \Phi.\ \mathcal{P}\ \varphi)$
 using *probability-unity*
 by (*induct n*, *auto*)

lemma (**in** *classical-logic*) *dirac-collapse*:
 $(\forall\ \mathcal{P} \in$ *probabilities*. $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
  $= (\forall\ \mathcal{P} \in$ *dirac-measures*. $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + \lceil c \rceil \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
proof
 assume $\forall\ \mathcal{P} \in$ *probabilities*. $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
 hence $\forall\ \mathcal{P} \in$ *dirac-measures*. $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$

```
    using dirac-measures-subset by fastforce
  thus ∀ 𝒫 ∈ dirac-measures. (∑ φ←Φ. 𝒫 φ) + ⌈c⌉ ≤ (∑ γ←Γ. 𝒫 γ)
    using dirac-ceiling by blast
next
  assume assm: ∀ 𝒫 ∈ dirac-measures. (∑ φ←Φ. 𝒫 φ) + ⌈c⌉ ≤ (∑ γ←Γ. 𝒫 γ)
  show ∀ 𝒫 ∈ probabilities. (∑ φ←Φ. 𝒫 φ) + c ≤ (∑ γ←Γ. 𝒫 γ)
  proof (cases c ≥ 0)
    case True
    from this obtain n :: nat where real n = ⌈c⌉
      by (metis (full-types)
            antisym-conv
            ceiling-le-zero
            ceiling-zero
            nat-0-iff
            nat-eq-iff2
            of-nat-nat)
    {
      fix 𝒫
      assume 𝒫 ∈ dirac-measures
      from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ 𝒫
        unfolding dirac-measures-def
        by auto
      have (∑ φ←Φ. 𝒫 φ) + ⌈c⌉ ≤ (∑ γ←Γ. 𝒫 γ)
        using assm ‹𝒫 ∈ dirac-measures› by blast
      hence (∑ φ←(replicate n ⊤) @ Φ. 𝒫 φ) ≤ (∑ γ←Γ. 𝒫 γ)
        using ‹real n = ⌈c⌉›
              probability-replicate-verum [where Φ=Φ and n=n]
        by metis
    }
    hence ∀ 𝒫 ∈ dirac-measures.
            (∑ φ←(replicate n ⊤) @ Φ. 𝒫 φ) ≤ (∑ γ←Γ. 𝒫 γ)
      by blast
    hence †: ∀ 𝒫 ∈ probabilities.
            (∑ φ←(replicate n ⊤) @ Φ. 𝒫 φ) ≤ (∑ γ←Γ. 𝒫 γ)
      using weakly-additive-completeness-collapse by blast
    {
      fix 𝒫
      assume 𝒫 ∈ probabilities
      from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ 𝒫
        unfolding probabilities-def
        by auto
      have (∑ φ←(replicate n ⊤) @ Φ. 𝒫 φ) ≤ (∑ γ←Γ. 𝒫 γ)
        using † ‹𝒫 ∈ probabilities› by blast
      hence (∑ φ←Φ. 𝒫 φ) + c ≤ (∑ γ←Γ. 𝒫 γ)
        using ‹real n = ⌈c⌉›
              probability-replicate-verum [where Φ=Φ and n=n]
        by linarith
    }
    then show ?thesis by blast
```

```
  next
    case False
    hence ⌈c⌉ ≤ 0 by auto
    from this obtain n :: nat where real n = - ⌈c⌉
      by (metis neg-0-le-iff-le of-nat-nat)
    {
      fix 𝒫
      assume 𝒫 ∈ dirac-measures
      from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ 𝒫
        unfolding dirac-measures-def
        by auto
      have (∑ φ←Φ. 𝒫 φ) + ⌈c⌉ ≤ (∑ γ←Γ. 𝒫 γ)
        using assm ‹𝒫 ∈ dirac-measures› by blast
      hence (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←(replicate n ⊤) @ Γ. 𝒫 γ)
        using ‹real n = - ⌈c⌉›
              probability-replicate-verum [where Φ=Γ and n=n]
        by linarith
    }
    hence ∀ 𝒫 ∈ dirac-measures.
            (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←(replicate n ⊤) @ Γ. 𝒫 γ)
      by blast
    hence ‡: ∀ 𝒫 ∈ probabilities.
            (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←(replicate n ⊤) @ Γ. 𝒫 γ)
      using weakly-additive-completeness-collapse by blast
    {
      fix 𝒫
      assume 𝒫 ∈ probabilities
      from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ 𝒫
        unfolding probabilities-def
        by auto
      have (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←(replicate n ⊤) @ Γ. 𝒫 γ)
        using ‡ ‹𝒫 ∈ probabilities› by blast
      hence (∑ φ←Φ. 𝒫 φ) + c ≤ (∑ γ←Γ. 𝒫 γ)
        using ‹real n = - ⌈c⌉›
              probability-replicate-verum [where Φ=Γ and n=n]
        by linarith
    }
    then show ?thesis by blast
  qed
qed

lemma (in classical-logic) dirac-strict-floor:
  ∀ 𝒫 ∈ dirac-measures.
      ((∑ φ←Φ. 𝒫 φ) + c < (∑ γ←Γ. 𝒫 γ))
        = ((∑ φ←Φ. 𝒫 φ) + ⌊c⌋ + 1 ≤ (∑ γ←Γ. 𝒫 γ))
proof
  fix 𝒫 :: 'a ⇒ real
  let ?𝒫' = (λ φ. ⌊ 𝒫 φ ⌋) :: 'a ⇒ int
  assume 𝒫 ∈ dirac-measures
```

```
  hence ∀ φ. P φ = ?P' φ
    unfolding dirac-measures-def
    by (metis (mono-tags, lifting)
          mem-Collect-eq
          of-int-0
          of-int-1
          of-int-floor-cancel)
  hence A: (∑ φ←Φ. P φ) = (∑ φ←Φ. ?P' φ)
    by (induct Φ, auto)
  have B: (∑ γ←Γ. P γ) = (∑ γ←Γ. ?P' γ)
    using ‹∀ φ. P φ = ?P' φ› by (induct Γ, auto)
  have ((∑ φ←Φ. P φ) + c < (∑ γ←Γ. P γ))
        = ((∑ φ←Φ. ?P' φ) + c < (∑ γ←Γ. ?P' γ))
    unfolding A B by auto
  also have ... = ((∑ φ←Φ. ?P' φ) + ⌊c⌋ + 1 ≤ (∑ γ←Γ. ?P' γ))
    by linarith
  finally show ((∑ φ←Φ. P φ) + c < (∑ γ←Γ. P γ)) =
               ((∑ φ←Φ. P φ) + ⌊c⌋ + 1 ≤ (∑ γ←Γ. P γ))
    using A B by linarith
qed

lemma (in classical-logic) strict-dirac-collapse:
    (∀ P ∈ probabilities. (∑ φ←Φ. P φ) + c < (∑ γ←Γ. P γ))
  = (∀ P ∈ dirac-measures. (∑ φ←Φ. P φ) + ⌊c⌋ + 1 ≤ (∑ γ←Γ. P γ))
proof
  assume ∀ P ∈ probabilities. (∑ φ←Φ. P φ) + c < (∑ γ←Γ. P γ)
  hence ∀ P ∈ dirac-measures. (∑ φ←Φ. P φ) + c < (∑ γ←Γ. P γ)
    using dirac-measures-subset by blast
  thus ∀ P ∈ dirac-measures. ((∑ φ←Φ. P φ) + ⌊c⌋ + 1 ≤ (∑ γ←Γ. P γ))
    using dirac-strict-floor by blast
next
  assume ∀ P ∈ dirac-measures. ((∑ φ←Φ. P φ) + ⌊c⌋ + 1 ≤ (∑ γ←Γ. P γ))
  moreover have ⌊c⌋ + 1 = ⌈ (⌊c⌋ + 1) :: real⌉
    by simp
  ultimately have ⋆:
    ∀ P ∈ probabilities. ((∑ φ←Φ. P φ) + ⌊c⌋ + 1 ≤ (∑ γ←Γ. P γ))
    using dirac-collapse [of Φ ⌊c⌋ + 1 Γ]
    by auto
  show ∀ P ∈ probabilities. ((∑ φ←Φ. P φ) + c < (∑ γ←Γ. P γ))
  proof
    fix P :: 'a ⇒ real
    assume P ∈ probabilities
    hence (∑ φ←Φ. P φ) + ⌊c⌋ + 1 ≤ (∑ γ←Γ. P γ)
      using ⋆ by auto
    thus (∑ φ←Φ. P φ) + c < (∑ γ←Γ. P γ)
      by linarith
  qed
qed
```

4.5 MaxSAT Completeness For Probability Logic

It follows from the collapse theorem that any probability inequality tautology, include those with *constant terms*, may be reduced to a bounded MaxSAT problem. This is not only a key computational complexity result, but suggests a straightforward algorithm for *computing* probability identities.

lemma (**in** *classical-logic*) *relative-maximals-verum-extract*:
assumes $\neg \vdash \varphi$
shows $(|\ \mathit{replicate}\ n\ \top\ @\ \Phi\ |_\varphi) = n + (|\ \Phi\ |_\varphi)$
proof (*induct n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
{
fix Φ
obtain Σ **where** $\Sigma \in \mathcal{M}\ (\top\ \#\ \Phi)\ \varphi$
using *assms relative-maximals-existence* **by** *fastforce*
hence $\top \in \mathit{set}\ \Sigma$
by (*metis* (*no-types*, *lifting*)
list.set-intros(*1*)
list-deduction-modus-ponens
list-deduction-weaken
relative-maximals-complement-equiv
relative-maximals-def
verum-tautology
mem-Collect-eq)
hence $\neg\ (\mathit{remove1}\ \top\ \Sigma :\vdash \varphi)$
by (*meson* $\langle\Sigma \in \mathcal{M}\ (\top\ \#\ \Phi)\ \varphi\rangle$
list.set-intros(*1*)
axiom-k
list-deduction-modus-ponens
list-deduction-monotonic
list-deduction-weaken
relative-maximals-complement-equiv
set-remove1-subset)
moreover
have *mset* $\Sigma \subseteq\#$ *mset* $(\top\ \#\ \Phi)$
using $\langle\Sigma \in \mathcal{M}\ (\top\ \#\ \Phi)\ \varphi\rangle$ *relative-maximals-def* **by** *blast*
hence *mset* $(\mathit{remove1}\ \top\ \Sigma) \subseteq\#$ *mset* Φ
using *subset-eq-diff-conv* **by** *fastforce*
ultimately have $(|\ \Phi\ |_\varphi) \geq \mathit{length}\ (\mathit{remove1}\ \top\ \Sigma)$
by (*metis* (*no-types*, *lifting*)
relative-MaxSAT-intro
list-deduction-weaken
relative-maximals-def
relative-maximals-existence

```
        mem-Collect-eq)
    hence (| Φ |_φ) + 1 ≥ length Σ
      by (simp add: ‹⊤ ∈ set Σ› length-remove1)
    moreover have (| Φ |_φ) < length Σ
    proof (rule ccontr)
      assume ¬ (| Φ |_φ) < length Σ
      hence (| Φ |_φ) ≥ length Σ by linarith
      from this obtain Δ where Δ ∈ M Φ φ length Δ ≥ length Σ
        using assms relative-MaxSAT-intro relative-maximals-existence by fastforce
      hence ¬ (⊤ # Δ) :⊢ φ
        using list-deduction-modus-ponens
              list-deduction-theorem
              list-deduction-weaken
              relative-maximals-def
              verum-tautology
        by blast
      moreover have mset (⊤ # Δ) ⊆# mset (⊤ # Φ)
        using ‹Δ ∈ M Φ φ› relative-maximals-def by auto
      ultimately have length Σ ≥ length (⊤ # Δ)
        using ‹Σ ∈ M (⊤ # Φ) φ› relative-maximals-def by blast
      hence length Δ ≥ length (⊤ # Δ)
        using ‹length Σ ≤ length Δ› dual-order.trans by blast
      thus False by simp
    qed
    ultimately have (| ⊤ # Φ |_φ) = (1 + | Φ |_φ)
     by (metis Suc-eq-plus1 Suc-le-eq ‹Σ ∈ M (⊤ # Φ) φ› add.commute le-antisym
relative-MaxSAT-intro)
  }
  thus ?case using Suc by simp
qed

lemma (in classical-logic) complement-MaxSAT-completeness:
  (∀ P ∈ dirac-measures. (∑ φ←Φ. P φ) ≤ (∑ γ←Γ. P γ)) = (length Φ ≤ ‖ ∼
Γ @ Φ ‖_⊥)
proof (cases ⊢ ⊥)
  case True
  hence M (∼ Γ @ Φ) ⊥ = {}
    using relative-maximals-existence by auto
  hence length (∼ Γ @ Φ) = ‖ ∼ Γ @ Φ ‖_⊥
   unfolding complement-relative-MaxSAT-def relative-MaxSAT-def by presburger
  then show ?thesis
   using True counting-deduction-completeness counting-deduction-tautology-weaken
    by auto
next
  case False
  then show ?thesis
   using counting-deduction-completeness relative-maximals-counting-deduction-lower-bound
    by blast
qed
```

```
lemma (in classical-logic) relative-maximals-neg-verum-elim:
  (| replicate n (∼ ⊤) @ Φ |_φ) = (| Φ |_φ)
proof (induct n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  {
    fix Φ
    have (| (∼ ⊤) # Φ |_φ) = (| Φ |_φ)
    proof (cases ⊢ φ)
      case True
      then show ?thesis
        unfolding relative-MaxSAT-def relative-maximals-def
        by (simp add: list-deduction-weaken)
    next
      case False
      from this obtain Σ where Σ ∈ 𝓜 ((∼ ⊤) # Φ) φ
        using relative-maximals-existence by fastforce
      have [(∼ ⊤)] :⊢ φ
        by (metis modus-ponens
                  Peirces-law
                  pseudo-scotus
                  list-deduction-theorem
                  list-deduction-weaken
                  negation-def
                  verum-def)
      hence ∼ ⊤ ∉ set Σ
        by (meson ‹Σ ∈ 𝓜 (∼ ⊤ # Φ) φ›
                  list.set-intros(1)
                  list-deduction-base-theory
                  list-deduction-theorem
                  list-deduction-weaken
                  relative-maximals-complement-equiv)
      hence remove1 (∼ ⊤) Σ = Σ
        by (simp add: remove1-idem)
      moreover have mset Σ ⊆# mset ((∼ ⊤) # Φ)
        using ‹Σ ∈ 𝓜 (∼ ⊤ # Φ) φ› relative-maximals-def by blast
      ultimately have mset Σ ⊆# mset Φ
     by (metis add-mset-add-single mset.simps(2) mset-remove1 subset-eq-diff-conv)
      moreover have ¬ (Σ :⊢ φ)
        using ‹Σ ∈ 𝓜 (∼ ⊤ # Φ) φ› relative-maximals-def by blast
      ultimately have (| Φ |_φ) ≥ length Σ
        by (metis (no-types, lifting)
                  relative-MaxSAT-intro
                  list-deduction-weaken
                  relative-maximals-def
                  relative-maximals-existence
```

```
            mem-Collect-eq)
    hence (| Φ |_φ) ≥ (| (∼ ⊤) # Φ |_φ)
      using ‹Σ ∈ M (∼ ⊤ # Φ) φ› relative-MaxSAT-intro by auto
    moreover
    have (| Φ |_φ) ≤ (| (∼ ⊤) # Φ |_φ)
    proof −
      obtain Δ where Δ ∈ M Φ φ
        using False relative-maximals-existence by blast
      hence
        ¬ Δ :⊢ φ
        mset Δ ⊆# mset ((∼ ⊤) # Φ)
        unfolding relative-maximals-def
        by (simp,
            metis (mono-tags, lifting)
                  Diff-eq-empty-iff-mset
                  list-subtract.simps(2)
                  list-subtract-mset-homomorphism
                  relative-maximals-def
                  mem-Collect-eq
                  mset-zero-iff
                  remove1.simps(1))
      hence length Δ ≤ length Σ
        using ‹Σ ∈ M (∼ ⊤ # Φ) φ› relative-maximals-def by blast
      thus ?thesis
        using ‹Δ ∈ M Φ φ› ‹Σ ∈ M (∼ ⊤ # Φ) φ› relative-MaxSAT-intro by
auto
    qed
    ultimately show ?thesis
      using le-antisym by blast
  qed
 }
 thus ?case using Suc by simp
qed

lemma (in classical-logic) dirac-MaxSAT-partial-completeness:
 (∀ P ∈ dirac-measures. (∑ φ←Φ. P φ) ≤ (∑ γ←Γ. P γ)) = (MaxSAT (∼ Γ @
Φ ) ≤ length Γ)
proof −
 {
  fix P :: 'a ⇒ real
  obtain ϱ :: 'a list ⇒ 'a list ⇒ 'a ⇒ real where
      (∀ Φ Γ. ϱ Φ Γ ∈ dirac-measures ∧ ¬ (∑ φ←Φ. (ϱ Φ Γ) φ) ≤ (∑ γ←Γ. (ϱ
Φ Γ) γ)
            ∨ length Φ ≤ ‖ ∼ Γ @ Φ ‖⊥)
    ∧ (∀ Φ Γ. length Φ ≤ (‖ ∼ Γ @ Φ ‖⊥)
            ⟶ (∀ P ∈ dirac-measures. (∑ φ←Φ. P φ) ≤ (∑ γ←Γ. P γ)))
  using complement-MaxSAT-completeness by moura
 moreover have ∀ Γ φ n. length Γ − n ≤ (‖ Γ ‖φ) ∨ (| Γ |φ) − n ≠ 0
  by (metis add-diff-cancel-right')
```

cancel-ab-semigroup-add-class.diff-right-commute
diff-is-0-eq length-MaxSAT-decomposition)
moreover have $\forall\ \Gamma\ \Phi\ n.\ \mathit{length}\ (\Gamma\ @\ \Phi) - n \leq \mathit{length}\ \Gamma \vee \mathit{length}\ \Phi - n \neq 0$
by *force*
ultimately have
$(\mathcal{P} \in \mathit{dirac\text{-}measures} \longrightarrow (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
$\wedge\ (|\sim \Gamma\ @\ \Phi\ |_\perp) \leq \mathit{length}\ (\sim \Gamma)$
$\vee\quad \neg\ (|\sim \Gamma\ @\ \Phi\ |_\perp) \leq \mathit{length}\ (\sim \Gamma)$
$\wedge\ (\exists \mathcal{P}.\ \mathcal{P} \in \mathit{dirac\text{-}measures} \wedge \neg\ (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
by (*metis* (*no-types*) *add-diff-cancel-left′*
add-diff-cancel-right′
diff-is-0-eq length-append
length-MaxSAT-decomposition)
}
then show *?thesis* **by** *auto*
qed

lemma (**in** *consistent-classical-logic*) *dirac-inequality-elim*:
fixes c :: *real*
assumes $\forall\ \mathcal{P} \in \mathit{dirac\text{-}measures}.\ (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
shows $(\mathit{MaxSAT}\ (\sim \Gamma\ @\ \Phi) + c \leq \mathit{length}\ \Gamma)$
proof (*cases* $c \geq 0$)
case *True*
from *this* **obtain** n :: *nat* **where** *real* $n = \lceil c \rceil$
by (*metis ceiling-mono ceiling-zero of-nat-nat*)
{
fix $\mathcal{P}$
assume $\mathcal{P} \in \mathit{dirac\text{-}measures}$
from *this* **interpret** *probability-logic* $(\lambda\ \varphi.\ \vdash \varphi)\ (\rightarrow)\ \perp\ \mathcal{P}$
unfolding *dirac-measures-def*
by *auto*
have $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + n \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
by (*metis assms* ‹$\mathcal{P} \in \mathit{dirac\text{-}measures}$› ‹*real* $n = \lceil c \rceil$› *dirac-ceiling*)
hence $(\sum \varphi{\leftarrow}(\mathit{replicate}\ n\ \top)\ @\ \Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma)$
using *probability-replicate-verum* [**where** $\Phi{=}\Phi$ **and** $n{=}n$]
by *metis*
}
hence $(|\sim \Gamma\ @\ \mathit{replicate}\ n\ \top\ @\ \Phi\ |_\perp) \leq \mathit{length}\ \Gamma$
using *dirac-MaxSAT-partial-completeness* **by** *blast*
moreover have $\mathit{mset}\ (\sim \Gamma\ @\ \mathit{replicate}\ n\ \top\ @\ \Phi) = \mathit{mset}\ (\mathit{replicate}\ n\ \top\ @ \sim \Gamma\ @\ \Phi)$
by *simp*
ultimately have $(|\ \mathit{replicate}\ n\ \top\ @ \sim \Gamma\ @\ \Phi\ |_\perp) \leq \mathit{length}\ \Gamma$
unfolding *relative-MaxSAT-def relative-maximals-def*
by *metis*
hence $(|\sim \Gamma\ @\ \Phi\ |_\perp) + \lceil c \rceil \leq \mathit{length}\ \Gamma$
using ‹*real* $n = \lceil c \rceil$› *consistency relative-maximals-verum-extract*
by *auto*
then show *?thesis* **by** *linarith*

```
next
  case False
  hence ⌈c⌉ ≤ 0 by auto
  from this obtain n :: nat where real n = - ⌈c⌉
    by (metis neg-0-le-iff-le of-nat-nat)
  {
    fix 𝒫
    assume 𝒫 ∈ dirac-measures
    from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ 𝒫
      unfolding dirac-measures-def
      by auto
    have (∑ φ←Φ. 𝒫 φ) + ⌈c⌉ ≤ (∑ γ←Γ. 𝒫 γ)
      using assms ‹𝒫 ∈ dirac-measures› dirac-ceiling
      by blast
    hence (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←Γ. 𝒫 γ) + n
      using ‹real n = - ⌈c⌉› by linarith
    hence (∑ φ←Φ. 𝒫 φ) ≤ (∑ γ←(replicate n ⊤) @ Γ. 𝒫 γ)
      using probability-replicate-verum [where Φ=Γ and n=n]
      by metis
  }
  hence (| ∼ (replicate n ⊤ @ Γ) @ Φ |⊥) ≤ length (replicate n ⊤ @ Γ)
    using dirac-MaxSAT-partial-completeness [where Φ=Φ and Γ=replicate n ⊤
@ Γ]
    by metis
  hence (| ∼ Γ @ Φ |⊥) ≤ n + length Γ
    by (simp add: relative-maximals-neg-verum-elim)
  then show ?thesis using ‹real n = - ⌈c⌉› by linarith
qed

lemma (in classical-logic) dirac-inequality-intro:
  fixes c :: real
  assumes MaxSAT (∼ Γ @ Φ) + c ≤ length Γ
  shows ∀ 𝒫 ∈ dirac-measures. (∑ φ←Φ. 𝒫 φ) + c ≤ (∑ γ←Γ. 𝒫 γ)
proof (cases ⊢ ⊥)
  assume ⊢ ⊥
  {
    fix 𝒫
    assume 𝒫 ∈ dirac-measures
    from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ 𝒫
      unfolding dirac-measures-def
      by auto
    have False
      using ‹⊢ ⊥› consistency by blast
  }
  then show ?thesis by blast
next
  assume ¬ ⊢ ⊥
  then show ?thesis
  proof (cases c ≥ 0)
```

```
    assume c ≥ 0
    from this obtain n :: nat where real n = ⌈c⌉
      by (metis ceiling-mono ceiling-zero of-nat-nat)
    hence n + (| ∼ Γ @ Φ |⊥) ≤ length Γ
      using assms by linarith
    hence (| replicate n ⊤ @ ∼ Γ @ Φ |⊥) ≤ length Γ
      by (simp add: ‹¬ ⊢ ⊥› relative-maximals-verum-extract)
    moreover have mset (replicate n ⊤ @ ∼ Γ @ Φ) = mset (∼ Γ @ replicate n
⊤ @ Φ)
      by simp
    ultimately have (| ∼ Γ @ replicate n ⊤ @ Φ |⊥) ≤ length Γ
      unfolding relative-MaxSAT-def relative-maximals-def
      by metis
   hence ∀ P ∈ dirac-measures. (∑ φ←(replicate n ⊤) @ Φ. P φ) ≤ (∑ γ←Γ. P
γ)
      using dirac-MaxSAT-partial-completeness by blast
    {
      fix P
      assume P ∈ dirac-measures
      from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ P
        unfolding dirac-measures-def
        by auto
      have (∑ φ←(replicate n ⊤) @ Φ. P φ) ≤ (∑ γ←Γ. P γ)
        using ‹P ∈ dirac-measures›
              ‹∀ P ∈ dirac-measures. (∑ φ←(replicate n ⊤) @ Φ. P φ) ≤ (∑ γ←Γ.
P γ)›
        by blast
      hence (∑ φ← Φ. P φ) + n ≤ (∑ γ←Γ. P γ)
        by (simp add: probability-replicate-verum)
      hence (∑ φ← Φ. P φ) + c ≤ (∑ γ←Γ. P γ)
        using ‹real n = real-of-int ⌈c⌉› by linarith
    }
    then show ?thesis by blast
  next
    assume ¬ (c ≥ 0)
    hence ⌈c⌉ ≤ 0 by auto
    from this obtain n :: nat where real n = − ⌈c⌉
      by (metis neg-0-le-iff-le of-nat-nat)
    hence (| ∼ Γ @ Φ |⊥) ≤ n + length Γ
      using assms by linarith
    hence (| ∼ (replicate n ⊤ @ Γ) @ Φ |⊥) ≤ length (replicate n ⊤ @ Γ)
      by (simp add: relative-maximals-neg-verum-elim)
    hence ∀ P ∈ dirac-measures.
            (∑ φ←Φ. P φ) ≤ (∑ γ←(replicate n ⊤) @ Γ. P γ)
      using dirac-MaxSAT-partial-completeness by blast
    {
      fix P
      assume P ∈ dirac-measures
      from this interpret probability-logic (λ φ. ⊢ φ) (→) ⊥ P
```

unfolding *dirac-measures-def*
by *auto*
have $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}(\mathit{replicate}\ n\ \top)\ @\ \Gamma.\ \mathcal{P}\ \gamma)$
using ‹$\mathcal{P} \in \mathit{dirac\text{-}measures}$›
‹$\forall\ \mathcal{P} \in \mathit{dirac\text{-}measures}.$
$(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) \leq (\sum \gamma{\leftarrow}(\mathit{replicate}\ n\ \top)\ @\ \Gamma.\ \mathcal{P}\ \gamma)$›
by *blast*
hence $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + \lceil c \rceil \leq (\sum \gamma{\leftarrow}\ \Gamma.\ \mathcal{P}\ \gamma)$
using ‹$\mathit{real}\ n = -\lceil c \rceil$› *probability-replicate-verum* **by** *auto*
hence $(\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\ \Gamma.\ \mathcal{P}\ \gamma)$
by *linarith*
}
then show *?thesis* **by** *blast*
qed
qed

lemma (**in** *consistent-classical-logic*) *dirac-inequality-equiv*:
$(\forall\ \delta \in \mathit{dirac\text{-}measures}.\ (\sum \varphi{\leftarrow}\Phi.\ \delta\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \delta\ \gamma))$
$= (\mathit{MaxSAT}\ (\sim \Gamma\ @\ \Phi) + (c :: \mathit{real}) \leq \mathit{length}\ \Gamma)$
using *dirac-inequality-elim dirac-inequality-intro consistency* **by** *auto*

theorem (**in** *consistent-classical-logic*) *probability-inequality-equiv*:
$(\forall\ \mathcal{P} \in \mathit{probabilities}.\ (\sum \varphi{\leftarrow}\Phi.\ \mathcal{P}\ \varphi) + c \leq (\sum \gamma{\leftarrow}\Gamma.\ \mathcal{P}\ \gamma))$
$= (\mathit{MaxSAT}\ (\sim \Gamma\ @\ \Phi) + (c :: \mathit{real}) \leq \mathit{length}\ \Gamma)$
unfolding *dirac-collapse*
using *dirac-inequality-equiv dirac-ceiling* **by** *blast*

no-notation *first-component* ($\mathfrak{A}$)
no-notation *second-component* ($\mathfrak{B}$)
no-notation *merge-witness* ($\mathfrak{J}$)
no-notation *X-witness* ($\mathfrak{X}$)
no-notation *X-component* ($\mathfrak{X}_\bullet$)
no-notation *Y-witness* ($\mathfrak{Y}$)
no-notation *Y-component* ($\mathfrak{Y}_\bullet$)
no-notation *submerge-witness* ($\mathfrak{E}$)
no-notation *recover-witness-A* ($\mathfrak{P}$)
no-notation *recover-complement-A* ($\mathfrak{P}^C$)
no-notation *recover-witness-B* ($\mathfrak{Q}$)
no-notation *relative-maximals* ($\mathcal{M}$)
no-notation *relative-MaxSAT* (| - |- [45])
no-notation *complement-relative-MaxSAT* (‖ - ‖- [45])
no-notation *MaxSAT-optimal-pre-witness* ($\mathfrak{V}$)
no-notation *MaxSAT-optimal-witness* ($\mathfrak{W}$)
no-notation *disjunction-MaxSAT-optimal-witness* ($\mathfrak{W}_\sqcup$)
no-notation *implication-MaxSAT-optimal-witness* ($\mathfrak{W}_\rightarrow$)
no-notation *MaxSAT-witness* ($\mathfrak{U}$)

notation *FuncSet.funcset* (**infixr** $\rightarrow$ *60*)

end

Bibliography

[1] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, Feb. 1976.