Class-based Classical Propositional Logic

Matthew Doty

March 17, 2025

Abstract

We formulate classical propositional logic as an axiom class. Our class represents a Hilbert-style proof system with the axioms $\vdash \varphi \to \psi \to \varphi$, $\vdash (\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi$, and $\vdash ((\varphi \to \bot) \to \bot) \to \varphi$ along with the rule *modus ponens* $\vdash \varphi \to \psi \Longrightarrow \vdash \varphi \Longrightarrow \vdash \psi$. In this axiom class we provide lemmas to obtain *Maximally Consistent Sets* via Zorn's lemma. We define the concrete classical propositional calculus inductively and show it instantiates our axiom class. We formulate the usual semantics for the propositional calculus and show strong soundness and completeness. We provide conventional definitions of the other logical connectives and prove various common identities. Finally, we show that the propositional calculus *embeds* into any logic in our axiom class.

Contents

Chapter 1

Implication Logic

theory *Implication-Logic*
  imports *Main*
begin

This theory presents the pure implicational fragment of intuitionistic logic. That is to say, this is the fragment of intuitionistic logic containing *implication only*, and no other connectives nor *falsum* (i.e., ⊥). We shall refer to this logic as *implication logic* in future discussion.

For further reference see [7].

1.1 Axiomatization

Implication logic can be given by the a Hilbert-style axiom system, following Troelstra and Schwichtenberg [6, §1.3.9, pg. 33].

class *implication-logic* =
  fixes *deduction* :: $'a \Rightarrow bool$ (‹⊢ -› [*60*] *55*)
  fixes *implication* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹→› *70*)
  assumes *axiom-k*: $\vdash \varphi \to \psi \to \varphi$
  assumes *axiom-s*: $\vdash (\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi$
  assumes *modus-ponens*: $\vdash \varphi \to \psi \Longrightarrow \vdash \varphi \Longrightarrow \vdash \psi$

1.2 Common Rules

lemma (**in** *implication-logic*) *trivial-implication*:
  $\vdash \varphi \to \varphi$
  by (*meson axiom-k axiom-s modus-ponens*)

lemma (**in** *implication-logic*) *flip-implication*:
  $\vdash (\varphi \to \psi \to \chi) \to \psi \to \varphi \to \chi$
  by (*meson axiom-k axiom-s modus-ponens*)

lemma (**in** *implication-logic*) *hypothetical-syllogism*:
$\vdash (\psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi$
by (*meson axiom-k axiom-s modus-ponens*)

lemma (**in** *implication-logic*) *flip-hypothetical-syllogism*:
$\vdash (\psi \to \varphi) \to (\varphi \to \chi) \to (\psi \to \chi)$
using *modus-ponens flip-implication hypothetical-syllogism* **by** *blast*

lemma (**in** *implication-logic*) *implication-absorption*:
$\vdash (\varphi \to \varphi \to \psi) \to \varphi \to \psi$
by (*meson axiom-k axiom-s modus-ponens*)

1.3 Lists of Assumptions

1.3.1 List Implication

Implication given a list of assumptions can be expressed recursively

primrec (**in** *implication-logic*)
list-implication :: *'a list* ⇒ *'a* ⇒ *'a* (**infix** ‹:→› *80*) **where**
$[] :\to \varphi = \varphi$
$| \ (\psi \# \Psi) :\to \varphi = \psi \to \Psi :\to \varphi$

1.3.2 Deduction From a List of Assumptions

Deduction from a list of assumptions can be expressed in terms of (:→).

definition (**in** *implication-logic*) *list-deduction* :: *'a list* ⇒ *'a* ⇒ *bool* (**infix** ‹:⊢› *60*)
where
$\Gamma :\vdash \varphi \equiv \vdash \Gamma :\to \varphi$

1.3.3 List Deduction as Implication Logic

The relation (:⊢) may naturally be interpreted as a *deduction* predicate for an instance of implication logic for a fixed list of assumptions Γ.

Analogues of the two axioms of implication logic can be naturally stated using list implication.

lemma (**in** *implication-logic*) *list-implication-axiom-k*:
$\vdash \varphi \to \Gamma :\to \varphi$
by (*induct* Γ, (*simp*, *meson axiom-k axiom-s modus-ponens*)+)

lemma (**in** *implication-logic*) *list-implication-axiom-s*:
$\vdash \Gamma :\to (\varphi \to \psi) \to \Gamma :\to \varphi \to \Gamma :\to \psi$
by (*induct* Γ,
(*simp*, *meson axiom-k axiom-s modus-ponens hypothetical-syllogism*)+)

The lemmas $\vdash \varphi \to \Gamma :\to \varphi$ and $\vdash \Gamma :\to (\varphi \to \psi) \to \Gamma :\to \varphi \to \Gamma :\to \psi$ jointly give rise to an interpretation of implication logic, where a list of assumptions Γ play the role of a *background theory* of $(:\vdash)$.

context *implication-logic* **begin**
interpretation *list-deduction-logic*:
  implication-logic $\lambda\ \varphi.\ \Gamma :\vdash \varphi\ (\to)$
proof qed
 (*meson*
  list-deduction-def
  axiom-k
  axiom-s
  modus-ponens
  list-implication-axiom-k
  list-implication-axiom-s)+
end

The following *weakening* rule can also be derived.

lemma (**in** *implication-logic*) *list-deduction-weaken*:
 $\vdash \varphi \Longrightarrow \Gamma :\vdash \varphi$
 unfolding *list-deduction-def*
 using *modus-ponens list-implication-axiom-k*
 by *blast*

In the case of the empty list, the converse may be established.

lemma (**in** *implication-logic*) *list-deduction-base-theory* [*simp*]:
 $[] :\vdash \varphi \equiv \vdash \varphi$
 unfolding *list-deduction-def*
 by *simp*

lemma (**in** *implication-logic*) *list-deduction-modus-ponens*:
 $\Gamma :\vdash \varphi \to \psi \Longrightarrow \Gamma :\vdash \varphi \Longrightarrow \Gamma :\vdash \psi$
 unfolding *list-deduction-def*
 using *modus-ponens list-implication-axiom-s*
 by *blast*

1.4 The Deduction Theorem

One result in the meta-theory of implication logic is the *deduction theorem*, which is a mechanism for moving antecedents back and forth from collections of assumptions.

To develop the deduction theorem, the following two lemmas generalize $\vdash (\varphi \to \psi \to \chi) \to \psi \to \varphi \to \chi$.

lemma (**in** *implication-logic*) *list-flip-implication1*:
 $\vdash (\varphi \# \Gamma) :\to \chi \to \Gamma :\to (\varphi \to \chi)$
 by (*induct* Γ,
   (*simp*,

```
      meson
        axiom-k
        axiom-s
        modus-ponens
        flip-implication
        hypothetical-syllogism)+)
```

lemma (**in** *implication-logic*) *list-flip-implication2*:
  $\vdash \Gamma :\to (\varphi \to \chi) \to (\varphi \# \Gamma) :\to \chi$
  by (*induct* Γ,
    (*simp*,
      meson
        axiom-k
        axiom-s
        modus-ponens
        flip-implication
        hypothetical-syllogism)+)

Together the two lemmas above suffice to prove a form of the deduction theorem:

theorem (**in** *implication-logic*) *list-deduction-theorem*:
  $(\varphi \# \Gamma) :\vdash \psi = \Gamma :\vdash \varphi \to \psi$
  unfolding *list-deduction-def*
  by (*metis modus-ponens list-flip-implication1 list-flip-implication2*)

1.5 Monotonic Growth in Deductive Power

In logic, for two sets of assumptions Φ and Ψ, if $\Psi \subseteq \Phi$ then the latter theory Φ is said to be *stronger* than former theory Ψ. In principle, anything a weaker theory can prove a stronger theory can prove. One way of saying this is that deductive power increases monotonically with as the set of underlying assumptions grow.

The monotonic growth of deductive power can be expressed as a meta-theorem in implication logic.

The lemma $\vdash \Gamma :\to (\varphi \to \chi) \to (\varphi \# \Gamma) :\to \chi$ presents a means of *introducing* assumptions into a list of assumptions when those assumptions have been arrived at by an implication. The next lemma presents a means of *discharging* those assumptions, which can be used in the monotonic growth theorem to be proved.

lemma (**in** *implication-logic*) *list-implication-removeAll*:
  $\vdash \Gamma :\to \psi \to (\textit{removeAll}\ \varphi\ \Gamma) :\to (\varphi \to \psi)$
proof −
  have $\forall\ \psi.\ \vdash \Gamma :\to \psi \to (\textit{removeAll}\ \varphi\ \Gamma) :\to (\varphi \to \psi)$
  proof(*induct* Γ)
    case *Nil*

```
    then show ?case by (simp, meson axiom-k)
  next
    case (Cons χ Γ)
    assume
      inductive-hypothesis: ∀ ψ. ⊢ Γ :→ ψ → removeAll φ Γ :→ (φ → ψ)
    moreover {
      assume φ ≠ χ
      with inductive-hypothesis
      have ∀ ψ. ⊢ (χ # Γ) :→ ψ → removeAll φ (χ # Γ) :→ (φ → ψ)
        by (simp, meson modus-ponens hypothetical-syllogism)
    }
    moreover {
      fix ψ
      assume φ-equals-χ: φ = χ
      moreover with inductive-hypothesis
      have ⊢ Γ :→ (χ → ψ) → removeAll φ (χ # Γ) :→ (φ → χ → ψ) by simp
      hence ⊢ Γ :→ (χ → ψ) → removeAll φ (χ # Γ) :→ (φ → ψ)
        by (metis
              calculation
              modus-ponens
              implication-absorption
              list-flip-implication1
              list-flip-implication2
              list-implication.simps(2))
      ultimately have ⊢ (χ # Γ) :→ ψ → removeAll φ (χ # Γ) :→ (φ → ψ)
        by (simp,
              metis
                modus-ponens
                hypothetical-syllogism
                list-flip-implication1
                list-implication.simps(2))
    }
    ultimately show ?case by simp
  qed
  thus ?thesis by blast
qed
```

From lemma above presents what is needed to prove that deductive power for lists is monotonic.

```
theorem (in implication-logic) list-implication-monotonic:
  set Σ ⊆ set Γ ⟹ ⊢ Σ :→ φ → Γ :→ φ
proof −
  assume set Σ ⊆ set Γ
  moreover have ∀ Σ φ. set Σ ⊆ set Γ ⟶ ⊢ Σ :→ φ → Γ :→ φ
  proof(induct Γ)
    case Nil
    then show ?case
      by (metis
            list-implication.simps(1)
```

```
          list-implication-axiom-k
          set-empty
          subset-empty)
  next
    case (Cons ψ Γ)
    assume
      inductive-hypothesis: ∀ Σ φ. set Σ ⊆ set Γ ⟶ ⊢ Σ :→ φ → Γ :→ φ
    {
      fix Σ
      fix φ
      assume Σ-subset-relation: set Σ ⊆ set (ψ # Γ)
      have ⊢ Σ :→ φ → (ψ # Γ) :→ φ
      proof −
        {
          assume set Σ ⊆ set Γ
          hence ?thesis
            by (metis
                  inductive-hypothesis
                  axiom-k modus-ponens
                  flip-implication
                  list-implication.simps(2))
        }
        moreover {
          let ?Δ = removeAll ψ Σ
          assume ¬ (set Σ ⊆ set Γ)
          hence set ?Δ ⊆ set Γ
            using Σ-subset-relation by auto
          hence ⊢ ?Δ :→ (ψ → φ) → Γ :→ (ψ → φ)
            using inductive-hypothesis by auto
          hence ⊢ ?Δ :→ (ψ → φ) → (ψ # Γ) :→ φ
            by (metis
                  modus-ponens
                  flip-implication
                  list-flip-implication2
                  list-implication.simps(2))
          moreover have ⊢ Σ :→ φ → ?Δ :→ (ψ → φ)
            by (simp add: local.list-implication-removeAll)
          ultimately have ?thesis
            using modus-ponens hypothetical-syllogism by blast
        }
        ultimately show ?thesis by blast
      qed
    }
    thus ?case by simp
  qed
  ultimately show ?thesis by simp
qed
```

A direct consequence is that deduction from lists of assumptions is monotonic as well:

theorem (**in** *implication-logic*) *list-deduction-monotonic*:
  $set\ \Sigma \subseteq set\ \Gamma \Longrightarrow \Sigma :\vdash \varphi \Longrightarrow \Gamma :\vdash \varphi$
  unfolding *list-deduction-def*
  using *modus-ponens list-implication-monotonic*
  by *blast*

1.6 The Deduction Theorem Revisited

The monotonic nature of deduction allows us to prove another form of the deduction theorem, where the assumption being discharged is completely removed from the list of assumptions.

theorem (**in** *implication-logic*) *alternate-list-deduction-theorem*:
  $(\varphi \# \Gamma) :\vdash \psi = (removeAll\ \varphi\ \Gamma) :\vdash \varphi \rightarrow \psi$
  by (*metis*
    list-deduction-def
    modus-ponens
    filter-is-subset
    list-deduction-monotonic
    list-deduction-theorem
    list-implication-removeAll
    removeAll.simps(*2*)
    removeAll-filter-not-eq)

1.7 Reflection

In logic the *reflection* principle sometimes refers to when a collection of assumptions can deduce any of its members. It is automatically derivable from $[\![set\ \Sigma \subseteq set\ \Gamma;\ \Sigma :\vdash \varphi]\!] \Longrightarrow \Gamma :\vdash \varphi$ among the other rules provided.

lemma (**in** *implication-logic*) *list-deduction-reflection*:
  $\varphi \in set\ \Gamma \Longrightarrow \Gamma :\vdash \varphi$
  by (*metis*
    list-deduction-def
    insert-subset
    list.simps(*15*)
    list-deduction-monotonic
    list-implication.simps(*2*)
    list-implication-axiom-k
    order-refl)

1.8 The Cut Rule

Cut is a rule commonly presented in sequent calculi, dating back to Gerhard Gentzen's *Investigations in Logical Deduction* (1935) [4]

The cut rule is not generally necessary in sequent calculi. It can often be shown that the rule can be eliminated without reducing the power of the underlying logic. However, as demonstrated by George Boolos' *Don't Eliminate Cut* (1984) [3], removing the rule can often lead to very inefficient proof systems.

Here the rule is presented just as a meta theorem.

```
theorem (in implication-logic) list-deduction-cut-rule:
  (φ # Γ) :⊢ ψ ⟹ Δ :⊢ φ ⟹ Γ @ Δ :⊢ ψ
  by (metis
        (no-types, lifting)
        Un-upper1
        Un-upper2
        list-deduction-modus-ponens
        list-deduction-monotonic
        list-deduction-theorem
        set-append)
```

The cut rule can also be strengthened to entire lists of propositions.

```
theorem (in implication-logic) strong-list-deduction-cut-rule:
    (Φ @ Γ) :⊢ ψ ⟹ ∀ φ ∈ set Φ. Δ :⊢ φ ⟹ Γ @ Δ :⊢ ψ
proof −
  have ∀ ψ. (Φ @ Γ :⊢ ψ ⟶ (∀ φ ∈ set Φ. Δ :⊢ φ) ⟶ Γ @ Δ :⊢ ψ)
    proof(induct Φ)
      case Nil
      then show ?case
        by (metis
              Un-iff
              append.left-neutral
              list-deduction-monotonic
              set-append
              subsetI)
    next
      case (Cons χ Φ) assume inductive-hypothesis:
        ∀ ψ. Φ @ Γ :⊢ ψ ⟶ (∀φ∈set Φ. Δ :⊢ φ) ⟶ Γ @ Δ :⊢ ψ
      {
        fix ψ χ
        assume (χ # Φ) @ Γ :⊢ ψ
        hence A: Φ @ Γ :⊢ χ → ψ using list-deduction-theorem by auto
        assume ∀φ ∈ set (χ # Φ). Δ :⊢ φ
        hence B: ∀ φ ∈ set Φ. Δ :⊢ φ
          and C: Δ :⊢ χ by auto
        from A B have Γ @ Δ :⊢ χ → ψ using inductive-hypothesis by blast
        with C have Γ @ Δ :⊢ ψ
          by (meson
                list.set-intros(1)
                list-deduction-cut-rule
                list-deduction-modus-ponens
                list-deduction-reflection)
```

    }
    thus *?case* **by** *simp*
  qed
  moreover assume $(\Phi\ @\ \Gamma)\ :\vdash\ \psi$
moreover assume $\forall\ \varphi \in set\ \Phi.\ \Delta\ :\vdash\ \varphi$
ultimately show *?thesis* **by** *blast*
qed

1.9 Sets of Assumptions

While deduction in terms of lists of assumptions is straight-forward to define, deduction (and the *deduction theorem*) is commonly given in terms of *sets* of propositions. This formulation is suited to establishing strong completeness theorems and compactness theorems.

The presentation of deduction from a set follows the presentation of list deduction given for $(:\vdash)$.

1.10 Definition of Deduction

Just as deduction from a list $(:\vdash)$ can be defined in terms of $(:\rightarrow)$, deduction from a *set* of assumptions can be expressed in terms of $(:\vdash)$.

definition (**in** *implication-logic*) *set-deduction* :: *'a set* ⇒ *'a* ⇒ *bool* (**infix** ‹⊩› *60*)
  where
    $\Gamma \Vdash \varphi \equiv \exists\ \Psi.\ set\ \Psi\ \subseteq \Gamma \wedge \Psi\ :\vdash\ \varphi$

1.10.1 Interpretation as Implication Logic

As in the case of $(:\vdash)$, the relation $(\Vdash)$ may be interpreted as *deduction* predicate for a fixed set of assumptions Γ.

The following lemma is given in order to establish this, which asserts that every implication logic tautology $\vdash \varphi$ is also a tautology for $\Gamma \Vdash \varphi$.

lemma (**in** *implication-logic*) *set-deduction-weaken*:
  $\vdash \varphi \Longrightarrow \Gamma \Vdash \varphi$
  using *list-deduction-base-theory set-deduction-def* **by** *fastforce*

In the case of the empty set, the converse may be established.

lemma (**in** *implication-logic*) *set-deduction-base-theory*:
  $\{\} \Vdash \varphi \equiv \vdash \varphi$
  using *list-deduction-base-theory set-deduction-def* **by** *auto*

Next, a form of *modus ponens* is provided for $(\Vdash)$.

lemma (**in** *implication-logic*) *set-deduction-modus-ponens*:

$\Gamma \Vdash \varphi \to \psi \Longrightarrow \Gamma \Vdash \varphi \Longrightarrow \Gamma \Vdash \psi$
proof −
assume $\Gamma \Vdash \varphi \to \psi$
then obtain Φ **where** A: *set* $\Phi \subseteq \Gamma$ **and** B: Φ :⊢ $\varphi \to \psi$
using *set-deduction-def* **by** *blast*
assume $\Gamma \Vdash \varphi$
then obtain Ψ **where** C: *set* $\Psi \subseteq \Gamma$ **and** D: Ψ :⊢ φ
using *set-deduction-def* **by** *blast*
from B D **have** Φ @ Ψ :⊢ ψ
using *list-deduction-cut-rule list-deduction-theorem* **by** *blast*
moreover from A C **have** *set* (Φ @ Ψ) $\subseteq \Gamma$ **by** *simp*
ultimately show *?thesis*
using *set-deduction-def* **by** *blast*
qed

context *implication-logic* **begin**
interpretation *set-deduction-logic*:
implication-logic λ φ. $\Gamma \Vdash \varphi$ $(\to)$
proof
fix φ ψ
show $\Gamma \Vdash \varphi \to \psi \to \varphi$ **by** (*metis axiom-k set-deduction-weaken*)
next
fix φ ψ χ
show $\Gamma \Vdash (\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi$
by (*metis axiom-s set-deduction-weaken*)
next
fix φ ψ
show $\Gamma \Vdash \varphi \to \psi \Longrightarrow \Gamma \Vdash \varphi \Longrightarrow \Gamma \Vdash \psi$
using *set-deduction-modus-ponens* **by** *metis*
qed
end

1.11 The Deduction Theorem

The next result gives the deduction theorem for ($\Vdash$).

theorem (**in** *implication-logic*) *set-deduction-theorem*:
insert φ $\Gamma \Vdash \psi = \Gamma \Vdash \varphi \to \psi$
proof −
have $\Gamma \Vdash \varphi \to \psi \Longrightarrow$ *insert* φ $\Gamma \Vdash \psi$
by (*metis*
set-deduction-def
insert-mono
list.simps(15)
list-deduction-theorem)
moreover {
assume *insert* φ $\Gamma \Vdash \psi$
then obtain Φ **where** *set* $\Phi \subseteq$ *insert* φ Γ **and** Φ :⊢ ψ
using *set-deduction-def* **by** *auto*

hence *set* (*removeAll* φ Φ) $\subseteq \Gamma$ **by** *auto*
moreover from ‹Φ :⊢ ψ› **have** *removeAll* φ Φ :⊢ $\varphi \to \psi$
using *modus-ponens list-implication-removeAll list-deduction-def*
by *blast*
ultimately have $\Gamma \Vdash \varphi \to \psi$
using *set-deduction-def* **by** *blast*
}
ultimately show *insert* φ $\Gamma \Vdash \psi = \Gamma \Vdash \varphi \to \psi$ **by** *metis*
qed

1.12 Monotonic Growth in Deductive Power

In contrast to the (:⊢) relation, the proof that the deductive power of ($\Vdash$) grows monotonically with its assumptions may be fully automated.

theorem *set-deduction-monotonic*:
$\Sigma \subseteq \Gamma \Longrightarrow \Sigma \Vdash \varphi \Longrightarrow \Gamma \Vdash \varphi$
by (*meson dual-order.trans set-deduction-def*)

1.13 The Deduction Theorem Revisited

As a consequence of the fact that $[\![\Sigma \subseteq \Gamma; \Sigma \Vdash \varphi]\!] \Longrightarrow \Gamma \Vdash \varphi$ is automatically provable, an alternate *deduction theorem* where the discharged assumption is completely removed from the set of assumptions is just a consequence of the more conventional *insert* φ $\Gamma \Vdash \psi = \Gamma \Vdash \varphi \to \psi$ rule and some basic set identities.

theorem (**in** *implication-logic*) *alternate-set-deduction-theorem*:
insert φ $\Gamma \Vdash \psi = \Gamma - \{\varphi\} \Vdash \varphi \to \psi$
by (*metis insert-Diff-single set-deduction-theorem*)

1.14 Reflection

Just as in the case of (:⊢), deduction from sets of assumptions makes true the *reflection principle* and is automatically provable.

theorem (**in** *implication-logic*) *set-deduction-reflection*:
$\varphi \in \Gamma \Longrightarrow \Gamma \Vdash \varphi$
by (*metis*
Set.set-insert
list-implication.simps(*1*)
list-implication-axiom-k
set-deduction-theorem
set-deduction-weaken)

1.15 The Cut Rule

The final principle of $(\Vdash)$ presented is the *cut rule*.

First, the weak form of the rule is established.

theorem (**in** *implication-logic*) *set-deduction-cut-rule*:
  insert φ $\Gamma \Vdash \psi \Longrightarrow \Delta \Vdash \varphi \Longrightarrow \Gamma \cup \Delta \Vdash \psi$
proof −
  assume *insert* φ $\Gamma \Vdash \psi$
  hence $\Gamma \Vdash \varphi \to \psi$ **using** *set-deduction-theorem* **by** *auto*
  hence $\Gamma \cup \Delta \Vdash \varphi \to \psi$ **using** *set-deduction-def* **by** *auto*
  moreover assume $\Delta \Vdash \varphi$
  hence $\Gamma \cup \Delta \Vdash \varphi$ **using** *set-deduction-def* **by** *auto*
  ultimately show *?thesis* **using** *set-deduction-modus-ponens* **by** *metis*
qed

Another lemma is shown next in order to establish the strong form of the cut rule. The lemma shows the existence of a *covering list* of assumptions Ψ in the event some set of assumptions Δ proves everything in a finite set of assumptions Φ.

lemma (**in** *implication-logic*) *finite-set-deduction-list-deduction*:
  assumes *finite* Φ
  and $\forall\ \varphi \in \Phi.\ \Delta \Vdash \varphi$
  shows $\exists \Psi.$ *set* $\Psi \subseteq \Delta \wedge (\forall \varphi \in \Phi.\ \Psi :\vdash \varphi)$
  using *assms*
proof(*induct* Φ *rule*: *finite-induct*)
  case *empty* **thus** *?case* **by** (*metis all-not-in-conv empty-subsetI set-empty*)
next
  case (*insert* χ Φ)
  assume $\forall \varphi \in \Phi.\ \Delta \Vdash \varphi \Longrightarrow \exists \Psi.$ *set* $\Psi \subseteq \Delta \wedge (\forall \varphi \in \Phi.\ \Psi :\vdash \varphi)$
    and $\forall \varphi \in$ *insert* χ $\Phi.\ \Delta \Vdash \varphi$
  hence $\exists \Psi.$ *set* $\Psi \subseteq \Delta \wedge (\forall \varphi{\in}\Phi.\ \Psi :\vdash \varphi)$ **and** $\Delta \Vdash \chi$ **by** *simp+*
  then obtain Ψ_1 Ψ_2 **where**
    set $(\Psi_1 @ \Psi_2) \subseteq \Delta$
    $\forall \varphi \in \Phi.\ \Psi_1 :\vdash \varphi$
    $\Psi_2 :\vdash \chi$
    using *set-deduction-def* **by** *auto*
  moreover from *this* **have** $\forall \varphi \in$ (*insert* χ Φ). $\Psi_1 @ \Psi_2 :\vdash \varphi$
    by (*metis*
        insert-iff
        le-sup-iff
        list-deduction-monotonic
        order-refl set-append)
  ultimately show *?case* **by** *blast*
qed

With $[\![$*finite* Φ; $\forall \varphi{\in}\Phi.\ \Delta \Vdash \varphi]\!] \Longrightarrow \exists \Psi.$ *set* $\Psi \subseteq \Delta \wedge (\forall \varphi{\in}\Phi.\ \Psi :\vdash \varphi)$ the strengthened form of the cut rule can be given.

```
theorem (in implication-logic) strong-set-deduction-cut-rule:
  assumes Φ ∪ Γ ⊩ ψ
  and ∀ φ ∈ Φ. Δ ⊩ φ
  shows Γ ∪ Δ ⊩ ψ
proof −
  obtain Σ where
    A: set Σ  ⊆ Φ ∪ Γ and
    B: Σ :⊢ ψ
    using assms(1) set-deduction-def
    by auto+
  obtain Φ′ Γ′ where
    C: set Φ′ = set Σ ∩ Φ and
    D: set Γ′ = set Σ ∩ Γ
    by (metis inf-sup-aci(1) inter-set-filter)+
  then have set (Φ′ @ Γ′) = set Σ using A by auto
  hence E: Φ′ @ Γ′ :⊢ ψ using B list-deduction-monotonic by blast
  hence ∀ φ ∈ set Φ′. Δ ⊩ φ using assms(2) C by auto
  from this obtain Δ′ where set Δ′ ⊆ Δ and ∀ φ ∈ set Φ′. Δ′ :⊢ φ
    using finite-set-deduction-list-deduction by blast
  with strong-list-deduction-cut-rule D E
  have set (Γ′ @ Δ′) ⊆ Γ ∪ Δ and Γ′ @ Δ′ :⊢ ψ by auto
  thus ?thesis using set-deduction-def by blast
qed
```

1.16 Maximally Consistent Sets For Implication Logic

Maximally Consistent Sets are a common construction for proving completeness of logical calculi. For a classic presentation, see Dirk van Dalen's *Logic and Structure* (2013, §1.5, pgs. 42–45) [8].

Maximally consistent sets will form the foundation of all of the model theory we will employ in this text. In fact, apart from classical logic semantics, conventional model theory will not be used at all.

The models we are centrally concerned are derived from maximally consistent sets. These include probability measures used in completeness theorems of probability logic found in §**??**, as well as arbitrage protection and trading strategies stipulated by our formulation of the *Dutch Book Theorem* we present in §**??**.

Since implication logic does not have *falsum*, consistency is defined relative to a formula φ.

```
definition (in implication-logic)
  formula-consistent :: 'a ⇒ 'a set ⇒ bool (‹-−consistent -› [100] 100)
  where
    [simp]: φ−consistent Γ ≡ ¬ (Γ ⊩ φ)
```

Since consistency is defined relative to some φ, *maximal consistency* is presented as asserting that either ψ or $\psi \to \varphi$ is in the consistent set Γ, for all ψ. This coincides with the traditional definition in classical logic when φ is *falsum*.

definition (**in** *implication-logic*)
formula-maximally-consistent-set-def :: *'a* ⇒ *'a set* ⇒ *bool* (‹-−*MCS* -› [*100*] *100*)
where
[*simp*]: φ−*MCS* $\Gamma \equiv$ (φ−*consistent* Γ) $\wedge$ ($\forall\ \psi.\ \psi \in \Gamma \vee (\psi \to \varphi) \in \Gamma$)

Every consistent set Γ may be extended to a maximally consistent set.

However, no assumption is made regarding the cardinality of the types of an instance of *implication-logic*.

As a result, typical proofs that assume a countable domain are not suitable. Our proof leverages *Zorn's lemma*.

lemma (**in** *implication-logic*) *formula-consistent-extension*:
assumes φ−*consistent* Γ
shows (φ−*consistent* (*insert* ψ Γ)) $\vee$ (φ−*consistent* (*insert* ($\psi \to \varphi$) Γ))
proof −
{
assume ¬ φ−*consistent insert* ψ Γ
hence $\Gamma \Vdash \psi \to \varphi$
using *set-deduction-theorem*
unfolding *formula-consistent-def*
by *simp*
hence φ−*consistent insert* ($\psi \to \varphi$) Γ
by (*metis Un-absorb assms formula-consistent-def set-deduction-cut-rule*)
}
thus *?thesis* **by** *blast*
qed

theorem (**in** *implication-logic*) *formula-maximally-consistent-extension*:
assumes φ−*consistent* Γ
shows $\exists\ \Omega$. (φ−*MCS* Ω) $\wedge$ $\Gamma \subseteq \Omega$
proof −
let *?*Γ*-extensions* = {Σ. (φ−*consistent* Σ) $\wedge$ $\Gamma \subseteq \Sigma$}
have $\exists\ \Omega \in$ *?*Γ*-extensions*. $\forall \Sigma \in$ *?*Γ*-extensions*. $\Omega \subseteq \Sigma \longrightarrow \Sigma = \Omega$
proof (*rule subset-Zorn*)
fix $\mathcal{C}$ :: *'a set set*
assume *subset-chain-*$\mathcal{C}$: *subset.chain ?*Γ*-extensions* $\mathcal{C}$
hence $\mathcal{C}$: $\forall\ \Sigma \in \mathcal{C}.\ \Gamma \subseteq \Sigma\ \forall\ \Sigma \in \mathcal{C}.$ φ−*consistent* Σ
unfolding *subset.chain-def*
by *blast+*
show $\exists\ \Omega \in$ *?*Γ*-extensions*. $\forall\ \Sigma \in \mathcal{C}.\ \Sigma \subseteq \Omega$
proof *cases*
assume $\mathcal{C}$ = {} **thus** *?thesis* **using** *assms* **by** *blast*
next

```
    let ?Ω = ⋃ 𝒞
    assume 𝒞 ≠ {}
    hence Γ ⊆ ?Ω by (simp add: 𝒞(1) less-eq-Sup)
    moreover have φ−consistent ?Ω
    proof −
      {
        assume ¬ φ−consistent ?Ω
        then obtain ω where ω:
          finite ω
          ω ⊆ ?Ω
          ¬ φ−consistent ω
          unfolding
            formula-consistent-def
            set-deduction-def
          by auto
        from ω(1) ω(2) have ∃ Σ ∈ 𝒞. ω ⊆ Σ
        proof (induct ω rule: finite-induct)
          case empty thus ?case using ‹𝒞 ≠ {}› by blast
        next
          case (insert ψ ω)
          from this obtain Σ1 Σ2 where
            Σ1:
                ω ⊆ Σ1
                Σ1 ∈ 𝒞
            and Σ2:
                ψ ∈ Σ2
                Σ2 ∈ 𝒞
            by auto
          hence Σ1 ⊆ Σ2 ∨ Σ2 ⊆ Σ1
            using subset-chain-𝒞
            unfolding subset.chain-def
            by blast
          hence (insert ψ ω) ⊆ Σ1 ∨ (insert ψ ω) ⊆ Σ2
            using Σ1 Σ2 by blast
          thus ?case using Σ1 Σ2 by blast
        qed
        hence ∃ Σ ∈ 𝒞. (φ−consistent Σ) ∧ ¬ (φ−consistent Σ)
          using 𝒞(2) ω(3)
          unfolding
            formula-consistent-def
            set-deduction-def
          by auto
        hence False by auto
      }
      thus ?thesis by blast
    qed
    ultimately show ?thesis by blast
  qed
qed
```

```
  then obtain Ω where Ω:
    Ω ∈ ?Γ-extensions
    ∀ Σ ∈ ?Γ-extensions. Ω ⊆ Σ ⟶ Σ = Ω
    by auto+
  {
    fix ψ
    have (φ−consistent insert ψ Ω) ∨ (φ−consistent insert (ψ → φ) Ω)
         Γ ⊆ insert ψ Ω
         Γ ⊆ insert (ψ → φ) Ω
      using Ω(1) formula-consistent-extension formula-consistent-def
      by auto
    hence insert ψ Ω ∈ ?Γ-extensions
            ∨ insert (ψ → φ) Ω ∈ ?Γ-extensions
      by blast
    hence ψ ∈ Ω ∨ (ψ → φ) ∈ Ω using Ω(2) by blast
  }
  thus ?thesis
    using Ω(1)
    unfolding formula-maximally-consistent-set-def-def
    by blast
qed
```

Finally, maximally consistent sets contain anything that can be deduced from them, and model a form of *modus ponens*.

```
lemma (in implication-logic) formula-maximally-consistent-set-def-reflection:
  φ−MCS Γ ⟹ ψ ∈ Γ = Γ ⊩ ψ
proof −
  assume φ−MCS Γ
  {
    assume Γ ⊩ ψ
    moreover from ‹φ−MCS Γ› have ψ ∈ Γ ∨ (ψ → φ) ∈ Γ ¬ Γ ⊩ φ
      unfolding
        formula-maximally-consistent-set-def-def
        formula-consistent-def
      by auto
    ultimately have ψ ∈ Γ
      using set-deduction-reflection set-deduction-modus-ponens
      by metis
  }
  thus ψ ∈ Γ = Γ ⊩ ψ
    using set-deduction-reflection
    by metis
qed

theorem (in implication-logic) formula-maximally-consistent-set-def-implication-elimination:
  assumes φ−MCS Ω
  shows (ψ → χ) ∈ Ω ⟹ ψ ∈ Ω ⟹ χ ∈ Ω
  using
    assms
```

formula-maximally-consistent-set-def-reflection
set-deduction-modus-ponens
by *blast*

This concludes our introduction to implication logic.

end

Chapter 2

Classical Propositional Logic

theory *Classical-Logic*
  imports *Implication-Logic*
begin

This theory presents *classical propositional logic*, which is classical logic without quantifiers.

2.1 Axiomatization

Classical propositional logic can be given by the following Hilbert-style axiom system. It is *implication-logic* extended with *falsum* and double negation.

class *classical-logic* = *implication-logic* +
  fixes *falsum* :: $'a$ (‹⊥›)
  assumes *double-negation*: $\vdash (((\varphi \to \bot) \to \bot) \to \varphi)$

In some cases it is useful to assume consistency as an axiom:

class *consistent-classical-logic* = *classical-logic* +
  assumes *consistency*: $\neg \vdash \bot$

2.2 Common Rules

There are many common tautologies in classical logic. Once we have established *completeness* in §3, we will be able to leverage Isabelle/HOL's automation for proving these elementary results.

In order to bootstrap completeness, we develop some common lemmas using classical deduction alone.

lemma (**in** *classical-logic*)
  ex-falso-quodlibet: $\vdash \bot \to \varphi$
  using *axiom-k double-negation modus-ponens hypothetical-syllogism*

by *blast*

lemma (**in** *classical-logic*)
Contraposition: $\vdash ((\varphi \to \bot) \to (\psi \to \bot)) \to \psi \to \varphi$
proof −
have $[\varphi \to \bot, \psi, (\varphi \to \bot) \to (\psi \to \bot)] :\vdash \bot$
using *flip-implication list-deduction-theorem list-implication.simps(1)*
unfolding *list-deduction-def*
by *presburger*
hence $[\psi, (\varphi \to \bot) \to (\psi \to \bot)] :\vdash (\varphi \to \bot) \to \bot$
using *list-deduction-theorem* **by** *blast*
hence $[\psi, (\varphi \to \bot) \to (\psi \to \bot)] :\vdash \varphi$
using *double-negation list-deduction-weaken list-deduction-modus-ponens*
by *blast*
thus *?thesis*
using *list-deduction-base-theory list-deduction-theorem* **by** *blast*
qed

lemma (**in** *classical-logic*)
double-negation-converse: $\vdash \varphi \to (\varphi \to \bot) \to \bot$
by (*meson axiom-k modus-ponens flip-implication*)

The following lemma is sometimes referred to as *The Principle of Pseudo-Scotus*[2].

lemma (**in** *classical-logic*)
pseudo-scotus: $\vdash (\varphi \to \bot) \to \varphi \to \psi$
using *ex-falso-quodlibet modus-ponens hypothetical-syllogism* **by** *blast*

Another popular lemma is attributed to Charles Sanders Peirce, and has come to be known as *Peirces Law*[5].

lemma (**in** *classical-logic*) *Peirces-law*:
$\vdash ((\varphi \to \psi) \to \varphi) \to \varphi$
proof −
have $[\varphi \to \bot, (\varphi \to \psi) \to \varphi] :\vdash \varphi \to \psi$
using
pseudo-scotus
list-deduction-theorem
list-deduction-weaken
by *blast*
hence $[\varphi \to \bot, (\varphi \to \psi) \to \varphi] :\vdash \varphi$
by (*meson*
list.set-intros(1)
list-deduction-reflection
list-deduction-modus-ponens
set-subset-Cons
subsetCE)
hence $[\varphi \to \bot, (\varphi \to \psi) \to \varphi] :\vdash \bot$
by (*meson*
list.set-intros(1)

list-deduction-modus-ponens
list-deduction-reflection)
hence $[(\varphi \to \psi) \to \varphi] :\vdash (\varphi \to \bot) \to \bot$
using *list-deduction-theorem* **by** *blast*
hence $[(\varphi \to \psi) \to \varphi] :\vdash \varphi$
using *double-negation*
list-deduction-modus-ponens
list-deduction-weaken
by *blast*
thus *?thesis*
using *list-deduction-def*
by *auto*
qed

lemma (**in** *classical-logic*) *excluded-middle-elimination*:
$\vdash (\varphi \to \psi) \to ((\varphi \to \bot) \to \psi) \to \psi$
proof −
let $?\Gamma = [\psi \to \bot, \varphi \to \psi, (\varphi \to \bot) \to \psi]$
have $?\Gamma :\vdash (\varphi \to \bot) \to \psi$
$?\Gamma :\vdash \psi \to \bot$
by (*simp add*: *list-deduction-reflection*)+
hence $?\Gamma :\vdash (\varphi \to \bot) \to \bot$
by (*meson*
flip-hypothetical-syllogism
list-deduction-base-theory
list-deduction-monotonic
list-deduction-theorem
set-subset-Cons)
hence $?\Gamma :\vdash \varphi$
using
double-negation
list-deduction-modus-ponens
list-deduction-weaken
by *blast*
hence $?\Gamma :\vdash \psi$
by (*meson*
list.set-intros(*1*)
list-deduction-modus-ponens
list-deduction-reflection
set-subset-Cons subsetCE)
hence $[\varphi \to \psi, (\varphi \to \bot) \to \psi] :\vdash \psi$
using
Peirces-law
list-deduction-modus-ponens
list-deduction-theorem
list-deduction-weaken
by *blast*
thus *?thesis*
unfolding *list-deduction-def*

by *simp*
qed

2.3 Maximally Consistent Sets For Classical Logic

Relativized maximally consistent sets were introduced in §1.16. Often this is exactly what we want in a proof. A completeness theorem typically starts by assuming φ is not provable, then finding a $\varphi{-}MCS\ \Gamma$ which gives rise to a model which does not make φ true.

A more conventional presentation says that Γ is maximally consistent if and only if $\neg\ \Gamma \Vdash \bot$ and $\forall\,\psi.\ \psi \in \Gamma \lor \psi \to \varphi \in \Gamma$. This conventional presentation will come up when formulating MaxSAT in §**??**. This in turn allows us to formulate MaxSAT completeness for probability inequalities in §**??**, and reduce checking if a strategy will always lose money or if it will always make money if matched to bounded MaxSAT as part of our proof of the *Dutch Book Theorem* in §**??** and §**??** respectively.

definition (**in** *classical-logic*)
  consistent :: *'a set* ⇒ *bool* **where**
    [*simp*]: *consistent* Γ ≡ ⊥−*consistent* Γ

definition (**in** *classical-logic*)
  maximally-consistent-set :: *'a set* ⇒ *bool* (‹*MCS*›) **where**
    [*simp*]: *MCS* Γ ≡ ⊥−*MCS* Γ

lemma (**in** *classical-logic*)
  formula-maximally-consistent-set-def-negation: φ−*MCS* Γ ⟹ φ → ⊥ ∈ Γ
proof −
  assume φ−*MCS* Γ
  {
    assume φ → ⊥ ∉ Γ
    hence (φ → ⊥) → φ ∈ Γ
      using ‹φ−*MCS* Γ›
      unfolding *formula-maximally-consistent-set-def-def*
      by *blast*
    hence Γ ⊩ (φ → ⊥) → φ
      using *set-deduction-reflection*
      by *simp*
    hence Γ ⊩ φ
      using
        Peirces-law
        set-deduction-modus-ponens
        set-deduction-weaken
      by *metis*
    hence *False*
      using ‹φ−*MCS* Γ›

```
      unfolding
        formula-maximally-consistent-set-def-def
        formula-consistent-def
      by simp
  }
  thus ?thesis by blast
qed
```

Relative maximal consistency and conventional maximal consistency in fact coincide in classical logic.

```
lemma (in classical-logic)
  formula-maximal-consistency: (∃ φ. φ−MCS Γ) = MCS Γ
proof −
  {
    fix φ
    have φ−MCS Γ ⟹ MCS Γ
    proof −
      assume φ−MCS Γ
      have consistent Γ
        using
          ‹φ−MCS Γ›
          ex-falso-quodlibet [where φ=φ]
          set-deduction-weaken [where Γ=Γ]
          set-deduction-modus-ponens
        unfolding
          formula-maximally-consistent-set-def-def
          consistent-def
          formula-consistent-def
        by metis
      moreover {
        fix ψ
        have ψ → ⊥ ∉ Γ ⟹ ψ ∈ Γ
        proof −
          assume ψ → ⊥ ∉ Γ
          hence (ψ → ⊥) → φ ∈ Γ
            using ‹φ−MCS Γ›
            unfolding formula-maximally-consistent-set-def-def
            by blast
          hence Γ ⊩ (ψ → ⊥) → φ
            using set-deduction-reflection
            by simp
          also have Γ ⊩ φ → ⊥
            using ‹φ−MCS Γ›
                formula-maximally-consistent-set-def-negation
                set-deduction-reflection
            by simp
          hence Γ ⊩ (ψ → ⊥) → ⊥
            using calculation
                hypothetical-syllogism
```

```
          [where φ=ψ → ⊥ and ψ=φ and χ=⊥]
         set-deduction-weaken
          [where Γ=Γ]
         set-deduction-modus-ponens
     by metis
    hence Γ ⊩ ψ
     using double-negation
          [where φ=ψ]
         set-deduction-weaken
          [where Γ=Γ]
         set-deduction-modus-ponens
     by metis
    thus ?thesis
     using ‹φ−MCS Γ›
         formula-maximally-consistent-set-def-reflection
     by blast
   qed
  }
  ultimately show ?thesis
   unfolding maximally-consistent-set-def
         formula-maximally-consistent-set-def-def
         formula-consistent-def
         consistent-def
   by blast
 qed
}
thus ?thesis
 unfolding maximally-consistent-set-def
 by metis
qed
```

Finally, classical logic allows us to strengthen $[\![\varphi{-}MCS\ \Omega;\ \psi \to \chi \in \Omega;\ \psi \in \Omega]\!] \Longrightarrow \chi \in \Omega$ to a biconditional.

```
lemma (in classical-logic)
 formula-maximally-consistent-set-def-implication:
 assumes φ−MCS Γ
 shows ψ → χ ∈ Γ = (ψ ∈ Γ ⟶ χ ∈ Γ)
proof −
 {
  assume hypothesis: ψ ∈ Γ ⟶ χ ∈ Γ
  {
   assume ψ ∉ Γ
   have ∀ψ. φ → ψ ∈ Γ
    by (meson assms
         formula-maximally-consistent-set-def-negation
         formula-maximally-consistent-set-def-implication-elimination
         formula-maximally-consistent-set-def-reflection
         pseudo-scotus set-deduction-weaken)
   then have ∀χ ψ. insert χ Γ ⊩ ψ ∨ χ → φ ∉ Γ
```

```
      by (meson assms
                 axiom-k
                 formula-maximally-consistent-set-def-reflection
                 set-deduction-modus-ponens
                 set-deduction-theorem
                 set-deduction-weaken)
    hence ψ → χ ∈ Γ
      by (meson ‹ψ ∉ Γ›
                 assms
                 formula-maximally-consistent-set-def-def
                 formula-maximally-consistent-set-def-reflection
                 set-deduction-theorem)
  }
  moreover {
    assume χ ∈ Γ
    hence ψ → χ ∈ Γ
      by (metis assms
                 calculation
                 insert-absorb
                 formula-maximally-consistent-set-def-reflection
                 set-deduction-theorem)
  }
  ultimately have  ψ → χ ∈ Γ using hypothesis by blast
 }
 thus ?thesis
  using assms
        formula-maximally-consistent-set-def-implication-elimination
  by metis
qed

end
```

Chapter 3

Classical Soundness and Completeness

theory *Classical-Logic-Completeness*
  imports *Classical-Logic*
begin

The following presents soundness completeness of the classical propositional calculus for propositional semantics. The classical propositional calculus is sometimes referred to as the *sentential calculus*. We give a concrete algebraic data type for propositional formulae in §3.1. We inductively define a logical judgement $\vdash_{prop}$ for these formulae. We also define the Tarski truth relation $\models_{prop}$ inductively, which we present in §3.3.

The most significant results here are the *embedding theorems*. These theorems show that the propositional calculus can be embedded in any logic extending *classical-logic*. These theorems are proved in §3.5.

3.1 Syntax

Here we provide the usual language for formulae in the propositional calculus. It contains *falsum* $\bot$, implication ($\rightarrow$), and a way of constructing *atomic* propositions $\lambda\ \varphi\ .\ \langle\ \varphi\ \rangle$. Defining the language is straight-forward using an algebraic data type.

datatype *'a classical-propositional-formula* =
    Falsum (‹⊥›)
  | *Proposition 'a* (‹⟨ - ⟩› [*45*])
  | *Implication*
      'a classical-propositional-formula
      'a classical-propositional-formula (**infixr** ‹→› *70*)

3.2 Propositional Calculus

In this section we recursively define what a proof is in the classical propositional calculus. We provide the familiar K and S axioms, as well as *double negation* and *modus ponens*.

named-theorems *classical-propositional-calculus*
  Rules for the Propositional Calculus

inductive *classical-propositional-calculus* ::
  'a classical-propositional-formula ⇒ *bool* (‹$\vdash_{prop}$ -› [*60*] *55*)
  where
    axiom-k [*classical-propositional-calculus*]:
      $\vdash_{prop}\ \varphi \to \psi \to \varphi$
  | *axiom-s* [*classical-propositional-calculus*]:
      $\vdash_{prop}\ (\varphi \to \psi \to \chi) \to (\varphi \to \psi) \to \varphi \to \chi$
  | *double-negation* [*classical-propositional-calculus*]:
      $\vdash_{prop}\ ((\varphi \to \bot) \to \bot) \to \varphi$
  | *modus-ponens* [*classical-propositional-calculus*]:
      $\vdash_{prop}\ \varphi \to \psi \Longrightarrow \vdash_{prop}\ \varphi \Longrightarrow \vdash_{prop}\ \psi$

Our proof system for our propositional calculus is trivially an instance of *classical-logic*. The introduction rules for $\vdash_{prop}$ naturally reflect the axioms of the classical logic axiom class.

instantiation *classical-propositional-formula*
  :: (*type*) *classical-logic*
begin
definition [*simp*]: $\bot = \bot$
definition [*simp*]: $\vdash \varphi = \vdash_{prop}\ \varphi$
definition [*simp*]: $\varphi \to \psi = \varphi \to \psi$
instance by *standard* (*simp add*: *classical-propositional-calculus*)+
end

3.3 Propositional Semantics

Below we give the typical definition of the Tarski truth relation $\models_{prop}$.

primrec *classical-propositional-semantics* ::
  'a set ⇒ *'a classical-propositional-formula* ⇒ *bool*
  (**infix** ‹$\models_{prop}$› *65*)
  where
      $\mathfrak{M} \models_{prop} \langle\ p\ \rangle = (p \in \mathfrak{M})$
    | $\mathfrak{M} \models_{prop} \varphi \to \psi = (\mathfrak{M} \models_{prop} \varphi \longrightarrow \mathfrak{M} \models_{prop} \psi)$
    | $\mathfrak{M} \models_{prop} \bot = False$

Soundness of our calculus for these semantics is trivial.

theorem *classical-propositional-calculus-soundness*:
  $\vdash_{prop}\ \varphi \Longrightarrow \mathfrak{M} \models_{prop} \varphi$
  by (*induct rule*: *classical-propositional-calculus.induct*, *simp*+)

3.4 Soundness and Completeness Proofs

definition *strong-classical-propositional-deduction* ::
 'a classical-propositional-formula set
  ⇒ *'a classical-propositional-formula* ⇒ *bool*
 (**infix** ‹$\Vdash_{prop}$› *65*)
 where
  [*simp*]: $\Gamma \Vdash_{prop} \varphi \equiv \Gamma \Vdash \varphi$

definition *strong-classical-propositional-tarski-truth* ::
 'a classical-propositional-formula set
  ⇒ *'a classical-propositional-formula* ⇒ *bool*
 (**infix** ‹$\Vdash\!\!\models_{prop}$› *65*)
 where
  [*simp*]: $\Gamma \Vdash\!\!\models_{prop} \varphi \equiv \forall\ \mathfrak{M}.(\forall\ \gamma \in \Gamma.\ \mathfrak{M} \models_{prop} \gamma) \longrightarrow \mathfrak{M} \models_{prop} \varphi$

definition *theory-propositions* ::
 'a classical-propositional-formula set ⇒ *'a set* (‹⦃ - ⦄› [*50*])
 where
  [*simp*]: ⦃ Γ ⦄ = $\{p\ .\ \Gamma \Vdash_{prop} \langle\ p\ \rangle\}$

Below we give the main lemma for completeness: the *truth lemma*. This proof connects the maximally consistent sets developed in §1.16 and §2.3 with the semantics given in §3.3.

All together, the technique we are using essentially follows the approach by Blackburn et al. [1, §4.2, pgs. 196-201].

lemma *truth-lemma*:
 assumes *MCS* Γ
 shows $\Gamma \Vdash_{prop} \varphi \equiv$ ⦃ Γ ⦄ $\models_{prop} \varphi$
proof (*induct* φ)
 case *Falsum*
 then show *?case* **using** *assms* **by** *auto*
next
 case (*Proposition x*)
 then show *?case* **by** *simp*
next
 case (*Implication ψ χ*)
 thus *?case*
  unfolding *strong-classical-propositional-deduction-def*
  by (*metis*
    assms
    maximally-consistent-set-def
    formula-maximally-consistent-set-def-implication
    classical-propositional-semantics.simps(*2*)
    implication-classical-propositional-formula-def
    set-deduction-modus-ponens
    set-deduction-reflection)
qed

Here the truth lemma above is combined with $\varphi-consistent\ \Gamma \Longrightarrow \exists\,\Omega.\ \varphi-MCS\ \Omega \wedge \Gamma \subseteq \Omega$ proven in §3.3. These theorems together give rise to strong completeness for the propositional calculus.

theorem *classical-propositional-calculus-strong-soundness-and-completeness*:
  $\Gamma \Vdash_{prop} \varphi = \Gamma \models_{prop} \varphi$
proof −
  have *soundness*: $\Gamma \Vdash_{prop} \varphi \Longrightarrow \Gamma \models_{prop} \varphi$
  proof −
    assume $\Gamma \Vdash_{prop} \varphi$
    from *this* **obtain** Γ' **where** Γ': *set* $\Gamma' \subseteq \Gamma$ $\Gamma' :\vdash \varphi$
    by (*simp add*: *set-deduction-def*, *blast*)
    {
      fix $\mathfrak{M}$
      assume $\forall\ \gamma \in \Gamma.\ \mathfrak{M} \models_{prop} \gamma$
      hence $\forall\ \gamma \in$ *set* $\Gamma'.\ \mathfrak{M} \models_{prop} \gamma$ **using** $\Gamma'(1)$ **by** *auto*
      hence $\forall\ \varphi.\ \Gamma' :\vdash \varphi \longrightarrow \mathfrak{M} \models_{prop} \varphi$
      proof (*induct* Γ')
        case *Nil*
        then show *?case*
          by (*simp add*:
              classical-propositional-calculus-soundness
              list-deduction-def)
      next
        case (*Cons* ψ Γ')
        thus *?case* **using** *list-deduction-theorem* **by** *fastforce*
      qed
      with $\Gamma'(2)$ **have** $\mathfrak{M} \models_{prop} \varphi$ **by** *blast*
    }
    thus $\Gamma \models_{prop} \varphi$
      using *strong-classical-propositional-tarski-truth-def* **by** *blast*
  qed
  have *completeness*: $\Gamma \models_{prop} \varphi \Longrightarrow \Gamma \Vdash_{prop} \varphi$
  proof (*erule contrapos-pp*)
    assume $\neg\ \Gamma \Vdash_{prop} \varphi$
    hence $\exists\ \mathfrak{M}.\ (\forall\ \gamma \in \Gamma.\ \mathfrak{M} \models_{prop} \gamma) \wedge \neg\ \mathfrak{M} \models_{prop} \varphi$
    proof −
      from ‹$\neg\ \Gamma \Vdash_{prop} \varphi$› **obtain** Ω **where** Ω: $\Gamma \subseteq \Omega$ $\varphi-MCS\ \Omega$
        by (*meson*
            formula-consistent-def
            formula-maximally-consistent-extension
            strong-classical-propositional-deduction-def)
      hence $(\varphi \rightarrow \bot) \in \Omega$
        using *formula-maximally-consistent-set-def-negation* **by** *blast*
      hence $\neg\ \{\!\!\{\ \Omega\ \}\!\!\} \models_{prop} \varphi$
        using Ω
            formula-consistent-def
            formula-maximal-consistency
            formula-maximally-consistent-set-def-def
            truth-lemma

unfolding *strong-classical-propositional-deduction-def*
by *blast*
moreover have $\forall\ \gamma \in \Gamma.\ \{\!\!\{\ \Omega\ \}\!\!\} \models_{prop} \gamma$
using
formula-maximal-consistency
truth-lemma
Ω
set-deduction-reflection
unfolding *strong-classical-propositional-deduction-def*
by *blast*
ultimately show *?thesis* **by** *auto*
qed
thus $\neg\ \Gamma \Vdash_{prop} \varphi$
unfolding *strong-classical-propositional-tarski-truth-def*
by *simp*
qed
from *soundness completeness* **show** $\Gamma \Vdash_{prop} \varphi = \Gamma \Vdash_{prop} \varphi$
by *linarith*
qed

For our applications in §sec:propositional-embedding, we will only need a weaker form of soundness and completeness rather than the stronger form proved above.

theorem *classical-propositional-calculus-soundness-and-completeness*:
$\vdash_{prop} \varphi = (\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} \varphi)$
using *classical-propositional-calculus-soundness* [**where** φ=φ]
classical-propositional-calculus-strong-soundness-and-completeness
[**where** φ=φ **and** Γ={}]
strong-classical-propositional-deduction-def
[**where** φ=φ **and** Γ={}]
strong-classical-propositional-tarski-truth-def
[**where** φ=φ **and** Γ={}]
deduction-classical-propositional-formula-def [**where** φ=φ]
set-deduction-base-theory [**where** φ=φ]
by *metis*

instantiation *classical-propositional-formula*
:: (*type*) *consistent-classical-logic*
begin
instance by *standard*
(*simp add*: *classical-propositional-calculus-soundness-and-completeness*)
end

3.5 Embedding Theorem For the Propositional Calculus

A recurring technique to prove theorems in logic moving forward is *embed* our theorem into the classical propositional calculus.

Using our embedding, we can leverage completeness to turn our problem into semantics and dispatch to Isabelle/HOL's classical theorem provers.

In future work we may make a tactic for this, but for now we just manually leverage the technique throughout our subsequent proofs.

primrec (**in** *classical-logic*)
  classical-propositional-formula-embedding
  :: *'a classical-propositional-formula* ⇒ *'a* (‹⦇ - ⦈› [*50*]) **where**
    ⦇ ⟨ p ⟩ ⦈ = p
  | ⦇ φ → ψ ⦈ = ⦇ φ ⦈ → ⦇ ψ ⦈
  | ⦇ ⊥ ⦈ = ⊥

theorem (**in** *classical-logic*) *propositional-calculus*:
  $\vdash_{prop} \varphi \Longrightarrow \vdash$ ⦇ φ ⦈
  by (*induct rule*: *classical-propositional-calculus.induct*,
    (*simp add*: *axiom-k axiom-s double-negation modus-ponens*)+)

The following theorem in particular shows that it suffices to prove theorems using classical semantics to prove theorems about the logic under investigation.

theorem (**in** *classical-logic*) *propositional-semantics*:
  $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} \varphi \Longrightarrow \vdash$ ⦇ φ ⦈
  by (*simp add*:
    classical-propositional-calculus-soundness-and-completeness
    propositional-calculus)

end

Chapter 4

List Utility Theorems

theory *List-Utilities*
  imports
    HOL−Combinatorics.List-Permutation
begin

Throughout our work it will be necessary to reuse common lemmas regarding lists and multisets. These results are proved in the following section and reused by subsequent lemmas and theorems.

4.1 Multisets

lemma *length-sub-mset*:
  assumes *mset* Ψ ⊆# *mset* Γ
    and *length* Ψ >= *length* Γ
  shows *mset* Ψ = *mset* Γ
  using *assms*
  by (*metis*
    append-Nil2
    append-eq-append-conv
    leD
    linorder-neqE-nat
    mset-le-perm-append
    perm-length
    size-mset
    size-mset-mono)

lemma *set-exclusion-mset-simplify*:
  assumes ¬ (∃ *ψ* ∈ *set* Ψ. *ψ* ∈ *set* Σ)
    and *mset* Ψ ⊆# *mset* (Σ @ Γ)
  shows *mset* Ψ ⊆# *mset* Γ
using *assms*
proof (*induct* Σ)
  case *Nil*
  then show *?case* **by** *simp*

next
  case (*Cons* σ Σ)
  then show *?case*
    by (*cases* $\sigma \in$ *set* Ψ,
        fastforce,
        metis
          add.commute
          add-mset-add-single
          diff-single-trivial
          in-multiset-in-set
          mset.simps(*2*)
          notin-set-remove1
          remove-hd
          subset-eq-diff-conv
          union-code
          append-Cons)
qed

lemma *image-mset-cons-homomorphism*:
  image-mset mset (*image-mset* ((#) φ) Φ) = *image-mset* ((+) {# φ #}) (*image-mset mset* Φ)
  by (*induct* Φ, *simp*+)

lemma *image-mset-append-homomorphism*:
  image-mset mset (*image-mset* ((@) Δ) Φ) = *image-mset* ((+) (*mset* Δ)) (*image-mset mset* Φ)
  by (*induct* Φ, *simp*+)

lemma *image-mset-add-collapse*:
  fixes *A B* :: *'a multiset*
  shows *image-mset* ((+) *A*) (*image-mset* ((+) *B*) *X*) = *image-mset* ((+) (*A* + *B*)) *X*
  by (*induct X*, *simp*, *simp*)

lemma *remove1-remdups-removeAll*: *remove1 x* (*remdups A*) = *remdups* (*removeAll x A*)
proof (*induct A*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons a A*)
  then show *?case*
    by (*cases a* = *x*, (*simp add*: *Cons*)+)
qed

lemma *mset-remdups*:
  assumes *mset A* = *mset B*
  shows *mset* (*remdups A*) = *mset* (*remdups B*)
proof −

```
  have ∀ B. mset A = mset B ⟶ mset (remdups A) = mset (remdups B)
  proof (induct A)
    case Nil
    then show ?case by simp
  next
    case (Cons a A)
    {
      fix B
      assume mset (a # A) = mset B
      hence mset A = mset (remove1 a B)
        by (metis add-mset-add-mset-same-iff
                  list.set-intros(1)
                  mset.simps(2)
                  mset-eq-setD
                  perm-remove)
      hence mset (remdups A) = mset (remdups (remove1 a B))
        using Cons.hyps by blast
      hence mset (remdups (a # (remdups A))) = mset (remdups (a # (remdups
(remove1 a B))))
        by (metis mset-eq-setD set-eq-iff-mset-remdups-eq list.simps(15))
      hence   mset (remdups (a # (removeAll a (remdups A))))
            = mset (remdups (a # (removeAll a (remdups (remove1 a B)))))
      by (metis insert-Diff-single list.set(2) set-eq-iff-mset-remdups-eq set-removeAll)
      hence   mset (remdups (a # (remdups (removeAll a A))))
            = mset (remdups (a # (remdups (removeAll a (remove1 a B)))))
      by (metis distinct-remdups distinct-remove1-removeAll remove1-remdups-removeAll)
      hence mset (remdups (remdups (a # A))) = mset (remdups (remdups (a #
(remove1 a B))))
        by (metis ‹mset A = mset (remove1 a B)›
                  list.set(2)
                  mset-eq-setD
                  set-eq-iff-mset-remdups-eq)
      hence mset (remdups (a # A)) = mset (remdups (a # (remove1 a B)))
        by (metis remdups-remdups)
      hence mset (remdups (a # A)) = mset (remdups B)
        using ‹mset (a # A) = mset B› mset-eq-setD set-eq-iff-mset-remdups-eq by
blast
    }
    then show ?case by simp
  qed
  thus ?thesis using assms by blast
qed

lemma mset-mset-map-snd-remdups:
  assumes mset (map mset A) = mset (map mset B)
  shows mset (map (mset ∘ (map snd) ∘ remdups) A) = mset (map (mset ∘ (map
snd) ∘ remdups) B)
proof −
  {
```

fix *B* :: ('*a* × '*b*) *list list*
fix *b* :: ('*a* × '*b*) *list*
assume *b* ∈ *set B*
hence *mset* (*map* (*mset* ∘ (*map snd*) ∘ *remdups*) (*b* # (*remove1 b B*)))
= *mset* (*map* (*mset* ∘ (*map snd*) ∘ *remdups*) *B*)
proof (*induct B*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons b' B*)
then show *?case*
by (*cases b* = *b'*, *simp*+)
qed
}
note ♢ = *this*
have
∀ *B* :: ('*a* × '*b*) *list list*.
mset (*map mset A*) = *mset* (*map mset B*)
⟶ *mset* (*map* (*mset* ∘ (*map snd*) ∘ *remdups*) *A*) = *mset* (*map* (*mset* ∘ (*map snd*) ∘ *remdups*) *B*)
proof (*induct A*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a A*)
{
fix *B*
assume ♠: *mset* (*map mset* (*a* # *A*)) = *mset* (*map mset B*)
hence *mset a* ∈# *mset* (*map mset B*)
by (*simp*,
metis ♠
image-set
list.set-intros(*1*)
list.simps(*9*)
mset-eq-setD)
from *this* **obtain** *b* **where** †:
b ∈ *set B*
mset a = *mset b*
by *auto*
with ♠ **have** *mset* (*map mset A*) = *mset* (*remove1* (*mset b*) (*map mset B*))
by (*simp add*: *union-single-eq-diff*)
moreover have *mset B* = *mset* (*b* # *remove1 b B*) **using** † **by** *simp*
hence *mset* (*map mset B*) = *mset* (*map mset* (*b* # (*remove1 b B*)))
by (*simp*,
metis image-mset-add-mset
mset.simps(*2*)
mset-remove1)
ultimately have *mset* (*map mset A*) = *mset* (*map mset* (*remove1 b B*))
by *simp*

```
      hence   mset (map (mset ∘ (map snd) ∘ remdups) A)
            = mset (map (mset ∘ (map snd) ∘ remdups) (remove1 b B))
        using Cons.hyps by blast
       moreover have (mset ∘ (map snd) ∘ remdups) a = (mset ∘ (map snd) ∘
remdups) b
        using †(2) mset-remdups by fastforce
      ultimately have
          mset (map (mset ∘ (map snd) ∘ remdups) (a # A))
         = mset (map (mset ∘ (map snd) ∘ remdups) (b # (remove1 b B)))
        by simp
      moreover have
          mset (map (mset ∘ (map snd) ∘ remdups) (b # (remove1 b B)))
         = mset (map (mset ∘ (map snd) ∘ remdups) B)
        using †(1) ◇ by blast
      ultimately have
          mset (map (mset ∘ (map snd) ∘ remdups) (a # A))
         = mset (map (mset ∘ (map snd) ∘ remdups) B)
        by simp
    }
    then show ?case by blast
  qed
  thus ?thesis using assms by blast
qed

lemma mset-remdups-append-msub:
  mset (remdups A) ⊆# mset (remdups (B @ A))
proof −
  have ∀ B. mset (remdups A) ⊆# mset (remdups (B @ A))
  proof (induct A)
    case Nil
    then show ?case by simp
  next
    case (Cons a A)
    {
      fix B
      have †: mset (remdups (B @ (a # A))) = mset (remdups (a # (B @ A)))
        by (induct B, simp+)
      have mset (remdups (a # A)) ⊆# mset (remdups (B @ (a # A)))
      proof (cases a ∈ set B ∧ a ∉ set A)
        case True
       hence †: mset (remove1 a (remdups (B @ A))) = mset (remdups ((removeAll
a B) @ A))
          by (simp add: remove1-remdups-removeAll)
        hence    (add-mset a (mset (remdups A)) ⊆# mset (remdups (B @ A)))
               = (mset (remdups A) ⊆# mset (remdups ((removeAll a B) @ A)))
          using True
          by (simp add: insert-subset-eq-iff)
        then show ?thesis
          by (metis † Cons True
```

```
                  Un-insert-right
                  list.set(2)
                  mset.simps(2)
                  mset-subset-eq-insertD
                  remdups.simps(2)
                  set-append
                  set-eq-iff-mset-remdups-eq
                  set-mset-mset set-remdups)
     next
       case False
       then show ?thesis using † Cons by simp
     qed
   }
   thus ?case by blast
 qed
 thus ?thesis by blast
qed
```

4.2 List Mapping

The following notation for permutations is slightly nicer when formatted in LaTeX.

```
notation perm (infix ‹⇌› 50)

lemma map-monotonic:
  assumes mset A ⊆# mset B
  shows mset (map f A) ⊆# mset (map f B)
  by (simp add: assms image-mset-subseteq-mono)

lemma perm-map-perm-list-exists:
  assumes A ⇌ map f B
  shows ∃ B'. A = map f B' ∧ B' ⇌ B
proof −
  have ∀ B. A ⇌ map f B ⟶ (∃ B'. A = map f B' ∧ B' ⇌ B)
  proof (induct A)
    case Nil
    then show ?case by simp
  next
    case (Cons a A)
    {
      fix B
      assume a # A ⇌ map f B
      from this obtain b where b:
        b ∈ set B
        f b = a
        by (metis
              (full-types)
              imageE
```

```
            list.set-intros(1)
            set-map
            set-mset-mset)
      hence A ⇌ (remove1 (f b) (map f B))
           B ⇌ b # remove1 b B
        by (metis
            ‹a # A ⇌ map f B›
            perm-remove-perm
            remove-hd,
           meson b(1) perm-remove)
      hence A ⇌ (map f (remove1 b B))
        by (metis (no-types)
            list.simps(9)
            mset-map
            mset-remove1
            remove-hd)
      from this obtain B′ where B′:
        A = map f B′
        B′ ⇌ (remove1 b B)
        using Cons.hyps by blast
      with b have a # A = map f (b # B′)
        by simp
      moreover have B ⇌ b # B′
        by (metis B′(2) ‹mset B = mset (b # remove1 b B)› mset.simps(2))
      ultimately have ∃B′. a # A = map f B′ ∧ B′ ⇌ B
        by (meson perm-sym)
    }
    thus ?case by blast
  qed
  with assms show ?thesis by blast
qed

lemma mset-sub-map-list-exists:
  assumes mset Φ ⊆# mset (map f Γ)
  shows ∃ Φ′. mset Φ′ ⊆# mset Γ ∧ Φ = (map f Φ′)
proof −
  have ∀ Φ. mset Φ ⊆# mset (map f Γ)
            ⟶ (∃ Φ′. mset Φ′ ⊆# mset Γ ∧ Φ = (map f Φ′))
  proof (induct Γ)
    case Nil
    then show ?case by simp
  next
    case (Cons γ Γ)
    {
      fix Φ
      assume mset Φ ⊆# mset (map f (γ # Γ))
      have ∃Φ′. mset Φ′ ⊆# mset (γ # Γ) ∧ Φ = map f Φ′
      proof cases
        assume f γ ∈ set Φ
```

hence $f\ \gamma\ \#\ (remove1\ (f\ \gamma)\ \Phi) \rightleftharpoons \Phi$
by *force*
with ‹$mset\ \Phi \subseteq\#\ mset\ (map\ f\ (\gamma\ \#\ \Gamma))$›
have $mset\ (remove1\ (f\ \gamma)\ \Phi) \subseteq\#\ mset\ (map\ f\ \Gamma)$
by (*metis*
insert-subset-eq-iff
list.simps(*9*)
mset.simps(*2*)
mset-remove1
remove-hd)
from *this Cons* **obtain** Φ' **where** Φ':
$mset\ \Phi' \subseteq\#\ mset\ \Gamma$
$remove1\ (f\ \gamma)\ \Phi = map\ f\ \Phi'$
by *blast*
hence $mset\ (\gamma\ \#\ \Phi') \subseteq\#\ mset\ (\gamma\ \#\ \Gamma)$
and $f\ \gamma\ \#\ (remove1\ (f\ \gamma)\ \Phi) = map\ f\ (\gamma\ \#\ \Phi')$
by *simp+*
hence $\Phi \rightleftharpoons map\ f\ (\gamma\ \#\ \Phi')$
using ‹$f\ \gamma \in set\ \Phi$› *perm-remove*
by *metis*
from *this* **obtain** Φ'' **where** Φ'':
$\Phi = map\ f\ \Phi''$
$\Phi'' \rightleftharpoons \gamma\ \#\ \Phi'$
using *perm-map-perm-list-exists*
by *blast*
hence $mset\ \Phi'' \subseteq\#\ mset\ (\gamma\ \#\ \Gamma)$
by (*metis* ‹$mset\ (\gamma\ \#\ \Phi') \subseteq\#\ mset\ (\gamma\ \#\ \Gamma)$›)
thus *?thesis* **using** Φ'' **by** *blast*
next
assume $f\ \gamma \notin set\ \Phi$
have $mset\ \Phi - \{\#f\ \gamma\#\} = mset\ \Phi$
by (*metis* (*no-types*)
‹$f\ \gamma \notin set\ \Phi$›
diff-single-trivial
set-mset-mset)
moreover
have $mset\ (map\ f\ (\gamma\ \#\ \Gamma))$
$= add\text{-}mset\ (f\ \gamma)\ (image\text{-}mset\ f\ (mset\ \Gamma))$
by *simp*
ultimately have $mset\ \Phi \subseteq\#\ mset\ (map\ f\ \Gamma)$
by (*metis* (*no-types*)
Diff-eq-empty-iff-mset
‹$mset\ \Phi \subseteq\#\ mset\ (map\ f\ (\gamma\ \#\ \Gamma))$›
add-mset-add-single
cancel-ab-semigroup-add-class.diff-right-commute
diff-diff-add mset-map)
with *Cons* **show** *?thesis*
by (*metis*
mset-le-perm-append

perm-append-single
subset-mset.order-trans)
qed
}
thus *?case* **using** *Cons* **by** *blast*
qed
thus *?thesis* **using** *assms* **by** *blast*
qed

4.3 Laws for Searching a List

lemma *find-Some-predicate*:
assumes *find P* Ψ = *Some* ψ
shows *P* ψ
using *assms*
proof (*induct* Ψ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* ω Ψ)
then show *?case* **by** (*cases P* ω, *fastforce*+)
qed

lemma *find-Some-set-membership*:
assumes *find P* Ψ = *Some* ψ
shows ψ ∈ *set* Ψ
using *assms*
proof (*induct* Ψ)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* ω Ψ)
then show *?case* **by** (*cases P* ω, *fastforce*+)
qed

4.4 Permutations

lemma *perm-count-list*:
assumes Φ ⇌ Ψ
shows *count-list* Φ φ = *count-list* Ψ φ
using *assms*
proof (*induct* Φ *arbitrary*: Ψ)
case *Nil*
then show *?case*
by *blast*
next
case (*Cons* χ Φ Ψ)
hence ◇: *count-list* Φ φ = *count-list* (*remove1* χ Ψ) φ

by (*metis mset-remove1 remove-hd*)
show *?case*
proof *cases*
assume $\chi = \varphi$
hence *count-list* $(\chi \# \Phi)$ φ = *count-list* Φ φ + *1* **by** *simp*
with ◇ **have** *count-list* $(\chi \# \Phi)$ φ = *count-list* (*remove1* χ Ψ) φ + *1*
by *simp*
moreover
have $\chi \in$ *set* Ψ
by (*metis Cons.prems list.set-intros*(*1*) *set-mset-mset*)
hence *count-list* (*remove1* χ Ψ) φ + *1* = *count-list* Ψ φ
using ‹$\chi = \varphi$›
by (*induct* Ψ, *simp*, *auto*)
ultimately show *?thesis* **by** *simp*
next
assume $\chi \neq \varphi$
with ◇ **have** *count-list* $(\chi \# \Phi)$ φ = *count-list* (*remove1* χ Ψ) φ
by *simp*
moreover have *count-list* (*remove1* χ Ψ) φ = *count-list* Ψ φ
using ‹$\chi \neq \varphi$›
by (*induct* Ψ, *simp*+)
ultimately show *?thesis* **by** *simp*
qed
qed

lemma *count-list-append*:
count-list (*A* @ *B*) *a* = *count-list A a* + *count-list B a*
by (*induct A*, *simp*, *simp*)

lemma *concat-remove1*:
assumes $\Psi \in$ *set* $\mathcal{L}$
shows *concat* $\mathcal{L}$ ⇌ Ψ @ *concat* (*remove1* Ψ $\mathcal{L}$)
using *assms*
by (*induct* $\mathcal{L}$, *simp*, *simp*, *metis*)

lemma *concat-set-membership-mset-containment*:
assumes *concat* Γ ⇌ Λ
and $\Phi \in$ *set* Γ
shows *mset* Φ ⊆# *mset* Λ
using *assms*
by (*induct* Γ, *simp*, *simp*, *metis concat-remove1 mset-le-perm-append*)

lemma (**in** *comm-monoid-add*) *perm-list-summation*:
assumes Ψ ⇌ Φ
shows $(\sum \psi' \leftarrow \Psi.\ f\ \psi') = (\sum \varphi' \leftarrow \Phi.\ f\ \varphi')$
using *assms*
proof (*induct* Ψ *arbitrary*: Φ)
case *Nil*
then show *?case* **by** *auto*

next
  case ($Cons\ \psi\ \Psi\ \Phi$)
  hence $(\sum \psi' \leftarrow \Psi.\ f\ \psi') = (\sum \varphi' \leftarrow (remove1\ \psi\ \Phi).\ f\ \varphi')$
    by (*metis mset-remove1 remove-hd*)
  moreover have $\psi \in set\ \Phi$
    by (*metis Cons.prems list.set-intros*(*1*) *set-mset-mset*)
  hence $(\sum \varphi' \leftarrow (\psi\ \#\ (remove1\ \psi\ \Phi)).\ f\ \varphi') = (\sum \varphi' \leftarrow \Phi.\ f\ \varphi')$
  proof (*induct* Φ)
    case *Nil*
    then show *?case* **by** *auto*
  next
    case ($Cons\ \varphi\ \Phi$)
    show *?case*
    proof *cases*
      assume $\varphi = \psi$
      then show *?thesis* **by** *simp*
    next
      assume $\varphi \neq \psi$
      hence $\psi \in set\ \Phi$
        using *Cons.prems* **by** *auto*
      hence $(\sum \varphi' \leftarrow (\psi\ \#\ (remove1\ \psi\ \Phi)).\ f\ \varphi') = (\sum \varphi' \leftarrow \Phi.\ f\ \varphi')$
        using *Cons.hyps* **by** *blast*
      hence $(\sum \varphi' \leftarrow (\varphi\ \#\ \Phi).\ f\ \varphi')$
              $= (\sum \varphi' \leftarrow (\psi\ \#\ \varphi\ \#\ (remove1\ \psi\ \Phi)).\ f\ \varphi')$
        by (*simp add*: *add.left-commute*)
      moreover
      have $(\psi\ \#\ (\varphi\ \#\ (remove1\ \psi\ \Phi))) = (\psi\ \#\ (remove1\ \psi\ (\varphi\ \#\ \Phi)))$
        using ‹$\varphi \neq \psi$› **by** *simp*
      ultimately show *?thesis*
        by *simp*
    qed
  qed
  ultimately show *?case*
    by *simp*
qed

4.5 List Duplicates

primrec *duplicates* :: *'a list* $\Rightarrow$ *'a set*
  where
    duplicates [] = {}
  | *duplicates* ($x\ \#\ xs$) =
      (*if* ($x \in set\ xs$)
       then insert x (*duplicates xs*)
       else duplicates xs)

lemma *duplicates-subset*:
  duplicates $\Phi \subseteq set\ \Phi$
  by (*induct* Φ, *simp*, *auto*)

```
lemma duplicates-alt-def:
  duplicates xs = {x. count-list xs x ≥ 2}
proof (induct xs)
  case Nil
  then show ?case by simp
next
  case (Cons x xs)
  assume inductive-hypothesis: duplicates xs = {x. 2 ≤ count-list xs x}
  then show ?case
  proof cases
    assume x ∈ set xs
    hence count-list (x # xs) x ≥ 2
      by (simp, induct xs, simp, simp, blast)
    hence {y. 2 ≤ count-list (x # xs) y}
             = insert x {y. 2 ≤ count-list xs y}
      by (simp, blast)
    thus ?thesis using inductive-hypothesis ‹x ∈ set xs›
      by simp
  next
    assume x ∉ set xs
    hence {y. 2 ≤ count-list (x # xs) y} = {y. 2 ≤ count-list xs y}
      by (simp, auto)
    thus ?thesis using inductive-hypothesis ‹x ∉ set xs›
      by simp
  qed
qed
```

4.6 List Subtraction

```
primrec list-subtract :: 'a list ⇒ 'a list ⇒ 'a list (infixl ‹⊖› 70)
  where
      xs ⊖ [] = xs
    | xs ⊖ (y # ys) = (remove1 y (xs ⊖ ys))

lemma list-subtract-mset-homomorphism [simp]:
  mset (A ⊖ B) = mset A − mset B
  by (induct B, simp, simp)

lemma list-subtract-empty [simp]:
  [] ⊖ Φ = []
  by (induct Φ, simp, simp)

lemma list-subtract-remove1-cons-perm:
  Φ ⊖ (φ # Λ) ⇌ (remove1 φ Φ) ⊖ Λ
  by (induct Λ, simp, simp add: add-mset-commute)

lemma list-subtract-cons:
  assumes φ ∉ set Λ
```

```
  shows (φ # Φ) ⊖ Λ = φ # (Φ ⊖ Λ)
  using assms
  by (induct Λ, simp, simp, blast)

lemma list-subtract-cons-absorb:
  assumes count-list Φ φ ≥ count-list Λ φ
  shows φ # (Φ ⊖ Λ) ⇌ (φ # Φ) ⊖ Λ
  using assms
proof (induct Λ arbitrary: Φ)
  case Nil
  then show ?case using list-subtract-cons by fastforce
next
  case (Cons ψ Λ Φ)
  then show ?case
  proof cases
    assume φ = ψ
    hence φ ∈ set Φ
      using Cons.prems count-notin by force
    hence Φ ⇌ φ # (remove1 ψ Φ)
      unfolding ‹φ = ψ›
      by force
    thus ?thesis using perm-count-list
      by (metis
            (no-types, lifting)
            Cons.hyps
            Cons.prems
            ‹φ = ψ›
            add-le-cancel-right
            add-mset-diff-bothsides
            count-list.simps(2)
            list-subtract-mset-homomorphism
            mset.simps(2))
  next
    assume φ ≠ ψ
    hence count-list (ψ # Λ) φ = count-list Λ φ
       by simp
    moreover have count-list Φ φ = count-list (remove1 ψ Φ) φ
    proof (induct Φ)
      case Nil
      then show ?case by simp
    next
      case (Cons φ' Φ)
      show ?case
      proof cases
        assume φ' = φ
        with ‹φ ≠ ψ›
        have count-list (φ' # Φ) φ = 1 + count-list Φ φ
             count-list (remove1 ψ (φ' # Φ)) φ
               = 1 + count-list (remove1 ψ Φ) φ
```

```
        by simp+
      with Cons show ?thesis by linarith
    next
      assume φ' ≠ φ
      with Cons show ?thesis by (cases φ' = ψ, simp+)
    qed
  qed
  ultimately show ?thesis
    using ‹count-list (ψ # Λ) φ ≤ count-list Φ φ›
    by (metis
          Cons.hyps
          ‹φ ≠ ψ›
          list-subtract-remove1-cons-perm
          mset.simps(2)
          remove1.simps(2))
 qed
qed

lemma list-subtract-cons-remove1-perm:
 assumes φ ∈ set Λ
 shows (φ # Φ) ⊖ Λ ⇌ Φ ⊖ (remove1 φ Λ)
 using assms
 by (metis
       list-subtract-mset-homomorphism
       list-subtract-remove1-cons-perm
       perm-remove
       remove-hd)

lemma list-subtract-removeAll-perm:
 assumes count-list Φ φ ≤ count-list Λ φ
 shows Φ ⊖ Λ ⇌ (removeAll φ Φ) ⊖ (removeAll φ Λ)
 using assms
proof (induct Φ arbitrary: Λ)
 case Nil
 then show ?case by auto
next
 case (Cons ξ Φ Λ)
 hence Φ ⊖ Λ ⇌ (removeAll φ Φ) ⊖ (removeAll φ Λ)
   by (metis add-leE count-list.simps(2))
 show ?case
 proof cases
   assume ξ = φ
   hence count-list Φ φ < count-list Λ φ
     using ‹count-list (ξ # Φ) φ ≤ count-list Λ φ›
     by auto
   hence count-list Φ φ ≤ count-list (remove1 φ Λ) φ
     by (induct Λ, simp, auto)
   hence Φ ⊖ (remove1 φ Λ)
           ⇌ removeAll φ Φ ⊖ removeAll φ (remove1 φ Λ)
```

using *Cons.hyps* **by** *blast*
hence $\Phi \ominus$ (*remove1* φ Λ) $\rightleftharpoons$ *removeAll* φ Φ $\ominus$ *removeAll* φ Λ
by (*simp add*: *filter-remove1 removeAll-filter-not-eq*)
moreover have $\varphi \in$ *set* Λ **and** $\varphi \in$ *set* (φ # Φ)
using ‹$\xi = \varphi$›
‹*count-list* (ξ # Φ) $\varphi \leq$ *count-list* Λ φ›
gr-implies-not0
by *fastforce+*
hence (φ # Φ) $\ominus$ Λ $\rightleftharpoons$ (*remove1* φ (φ # Φ)) $\ominus$ (*remove1* φ Λ)
by (*metis list-subtract-cons-remove1-perm remove-hd*)

hence (φ # Φ) $\ominus$ Λ $\rightleftharpoons$ Φ $\ominus$ (*remove1* φ Λ) **by** *simp*
ultimately show *?thesis* **using** ‹$\xi = \varphi$› **by** *auto*
next
assume $\xi \neq \varphi$
show *?thesis*
proof *cases*
assume $\xi \in$ *set* Λ
hence (ξ # Φ) $\ominus$ Λ $\rightleftharpoons$ Φ $\ominus$ *remove1* ξ Λ
by (*meson list-subtract-cons-remove1-perm*)
moreover have *count-list* Λ φ = *count-list* (*remove1* ξ Λ) φ
by (*metis*
count-list.simps(*2*)
‹$\xi \neq \varphi$›
‹$\xi \in$ *set* Λ›
perm-count-list
perm-remove)
hence *count-list* Φ $\varphi \leq$ *count-list* (*remove1* ξ Λ) φ
using ‹$\xi \neq \varphi$› ‹*count-list* (ξ # Φ) $\varphi \leq$ *count-list* Λ φ› **by** *auto*
hence Φ $\ominus$ *remove1* ξ Λ
$\rightleftharpoons$ (*removeAll* φ Φ) $\ominus$ (*removeAll* φ (*remove1* ξ Λ))
using *Cons.hyps* **by** *blast*
moreover
have (*removeAll* φ Φ) $\ominus$ (*removeAll* φ (*remove1* ξ Λ)) $\rightleftharpoons$
(*removeAll* φ Φ) $\ominus$ (*remove1* ξ (*removeAll* φ Λ))
by (*induct* Λ,
simp,
metis
‹$\xi \neq \varphi$›
list-subtract.simps(*2*)
mset-remove1
remove1.simps(*2*)
removeAll.simps(*2*))
hence (*removeAll* φ Φ) $\ominus$ (*removeAll* φ (*remove1* ξ Λ)) $\rightleftharpoons$
(*removeAll* φ (ξ # Φ)) $\ominus$ (*removeAll* φ Λ)
by (*metis*
‹$\xi \in$ *set* Λ›
‹$\xi \neq \varphi$›
list-subtract-cons-remove1-perm

```
          member-remove removeAll.simps(2)
          remove-code(1))
      ultimately show ?thesis
        by presburger
    next
      assume ξ ∉ set Λ
      hence (ξ # Φ) ⊖ Λ ⇌ ξ # (Φ ⊖ Λ)
        by fastforce
      hence (ξ # Φ) ⊖ Λ ⇌ ξ # ((removeAll φ Φ) ⊖ (removeAll φ Λ))
        using ‹Φ ⊖ Λ ⇌ removeAll φ Φ ⊖ removeAll φ Λ›
        by simp
      hence (ξ # Φ) ⊖ Λ ⇌ (ξ # (removeAll φ Φ)) ⊖ (removeAll φ Λ)
        by (simp add: ‹ξ ∉ set Λ› list-subtract-cons)
      thus ?thesis using ‹ξ ≠ φ› by auto
    qed
  qed
qed

lemma list-subtract-permute:
  assumes Φ ⇌ Ψ
  shows Φ ⊖ Λ ⇌ Ψ ⊖ Λ
  using assms
  by simp

lemma append-perm-list-subtract-intro:
  assumes A ⇌ B @ C
  shows A ⊖ C ⇌ B
proof −
  from ‹A ⇌ B @ C› have mset (A ⊖ C) = mset B
    by simp
  thus ?thesis by blast
qed

lemma list-subtract-concat:
  assumes Ψ ∈ set ℒ
  shows concat (ℒ ⊖ [Ψ]) ⇌ (concat ℒ) ⊖ Ψ
  using assms
  by (simp add: concat-remove1)

lemma (in comm-monoid-add) listSubstract-multisubset-list-summation:
  assumes mset Ψ ⊆# mset Φ
  shows (∑ψ←Ψ. f ψ) + (∑φ'←(Φ ⊖ Ψ). f φ') = (∑φ'←Φ. f φ')
proof −
  have ∀ Φ. mset Ψ ⊆# mset Φ
          ⟶ (∑ψ'←Ψ. f ψ') + (∑φ'←(Φ ⊖ Ψ). f φ') = (∑φ'←Φ. f φ')
  proof(induct Ψ)
    case Nil
    then show ?case
      by simp
```

next
  case (*Cons* ψ Ψ)
  {
    fix Φ
    assume *hypothesis*: *mset* (ψ # Ψ) $\subseteq$# *mset* Φ
    hence *mset* Ψ $\subseteq$# *mset* (*remove1* ψ Φ)
      by (*metis append-Cons mset-le-perm-append perm-remove-perm remove-hd*)
    hence
      $(\sum \psi' \leftarrow \Psi.\ f\ \psi') + (\sum \varphi' \leftarrow ((\mathit{remove1}\ \psi\ \Phi) \ominus \Psi).\ f\ \varphi')$
        $= (\sum \varphi' \leftarrow (\mathit{remove1}\ \psi\ \Phi).\ f\ \varphi')$
      using *Cons.hyps* **by** *blast*
    moreover have $(\mathit{remove1}\ \psi\ \Phi) \ominus \Psi \rightleftharpoons \Phi \ominus (\psi\ \#\ \Psi)$
      by (*meson list-subtract-remove1-cons-perm perm-sym*)
    hence $(\sum \varphi' \leftarrow ((\mathit{remove1}\ \psi\ \Phi) \ominus \Psi).\ f\ \varphi') = (\sum \varphi' \leftarrow (\Phi \ominus (\psi\ \#\ \Psi)).\ f\ \varphi')$
      using *perm-list-summation* **by** *blast*
    ultimately have
      $(\sum \psi' \leftarrow \Psi.\ f\ \psi') + (\sum \varphi' \leftarrow (\Phi \ominus (\psi\ \#\ \Psi)).\ f\ \varphi')$
        $= (\sum \varphi' \leftarrow (\mathit{remove1}\ \psi\ \Phi).\ f\ \varphi')$
      by *simp*
    hence
      $(\sum \psi' \leftarrow (\psi\ \#\ \Psi).\ f\ \psi') + (\sum \varphi' \leftarrow (\Phi \ominus (\psi\ \#\ \Psi)).\ f\ \varphi')$
        $= (\sum \varphi' \leftarrow (\psi\ \#\ (\mathit{remove1}\ \psi\ \Phi)).\ f\ \varphi')$
      by (*simp add*: *add.assoc*)
    moreover have $\psi \in$ *set* Φ
      by (*metis*
          append-Cons
          hypothesis
          list.set-intros(*1*)
          mset-le-perm-append
          perm-set-eq)
    hence $(\psi\ \#\ (\mathit{remove1}\ \psi\ \Phi)) \rightleftharpoons \Phi$
      by *auto*
    hence $(\sum \varphi' \leftarrow (\psi\ \#\ (\mathit{remove1}\ \psi\ \Phi)).\ f\ \varphi') = (\sum \varphi' \leftarrow \Phi.\ f\ \varphi')$
      using *perm-list-summation* **by** *blast*
    ultimately have
      $(\sum \psi' \leftarrow (\psi\ \#\ \Psi).\ f\ \psi') + (\sum \varphi' \leftarrow (\Phi \ominus (\psi\ \#\ \Psi)).\ f\ \varphi')$
        $= (\sum \varphi' \leftarrow \Phi.\ f\ \varphi')$
      by *simp*
  }
  then show *?case*
    by *blast*
qed
with *assms* **show** *?thesis* **by** *blast*
qed

lemma *list-subtract-set-difference-lower-bound*:
  set Γ $-$ *set* Φ $\subseteq$ *set* ($\Gamma \ominus \Phi$)
  using *subset-Diff-insert*
  by (*induct* Φ, *simp*, *fastforce*)

```
lemma list-subtract-set-trivial-upper-bound:
  set (Γ ⊖ Φ) ⊆ set Γ
      by (induct Φ,
          simp,
          simp,
          meson
            dual-order.trans
            set-remove1-subset)

lemma list-subtract-msub-eq:
  assumes mset Φ ⊆# mset Γ
      and length (Γ ⊖ Φ) = m
    shows length Γ = m + length Φ
  using assms
proof −
  have ∀ Γ. mset Φ ⊆# mset Γ
          ⟶ length (Γ ⊖ Φ) = m --> length Γ = m + length Φ
  proof (induct Φ)
    case Nil
    then show ?case by simp
  next
    case (Cons φ Φ)
    {
      fix Γ :: 'a list
      assume mset (φ # Φ) ⊆# mset Γ
             length (Γ ⊖ (φ # Φ)) = m
      moreover from this have
        mset Φ ⊆# mset (remove1 φ Γ)
        mset (Γ ⊖ (φ # Φ)) = mset ((remove1 φ Γ) ⊖ Φ)
        by (metis
              append-Cons
              mset-le-perm-append
              perm-remove-perm
              remove-hd,
            simp)
      ultimately have length (remove1 φ Γ) = m + length Φ
        using Cons.hyps
        by (metis mset-eq-length)
      hence length (φ # (remove1 φ Γ)) = m + length (φ # Φ)
        by simp
      moreover have φ ∈ set Γ
        by (metis
              ‹mset (Γ ⊖ (φ # Φ)) = mset (remove1 φ Γ ⊖ Φ)›
              ‹mset (φ # Φ) ⊆# mset Γ›
              ‹mset Φ ⊆# mset (remove1 φ Γ)›
              add-diff-cancel-left'
              add-right-cancel
              eq-iff
```

```
          impossible-Cons
          list-subtract-mset-homomorphism
          mset-subset-eq-exists-conv
          remove1-idem size-mset)
      hence length (φ # (remove1 φ Γ)) = length Γ
        by (metis
          One-nat-def
          Suc-pred
          length-Cons
          length-pos-if-in-set
          length-remove1)
      ultimately have length Γ = m + length (φ # Φ) by simp
    }
    thus ?case by blast
  qed
  thus ?thesis using assms by blast
qed

lemma list-subtract-not-member:
  assumes b ∉ set A
  shows A ⊖ B = A ⊖ (remove1 b B)
  using assms
  by (induct B,
      simp,
      simp,
      metis
        add-mset-add-single
        diff-subset-eq-self
        insert-DiffM2
        insert-subset-eq-iff
        list-subtract-mset-homomorphism
        remove1-idem
        set-mset-mset)

lemma list-subtract-monotonic:
  assumes mset A ⊆# mset B
  shows mset (A ⊖ C) ⊆# mset (B ⊖ C)
  by (simp,
      meson
        assms
        subset-eq-diff-conv
        subset-mset.dual-order.refl
        subset-mset.order-trans)

lemma map-list-subtract-mset-containment:
  mset ((map f A) ⊖ (map f B)) ⊆# mset (map f (A ⊖ B))
  by (induct B, simp, simp,
      metis
        diff-subset-eq-self
```

```
        diff-zero
        image-mset-add-mset
        image-mset-subseteq-mono
        image-mset-union
        subset-eq-diff-conv
        subset-eq-diff-conv)

lemma map-list-subtract-mset-equivalence:
  assumes mset B ⊆# mset A
  shows mset ((map f A) ⊖ (map f B)) = mset (map f (A ⊖ B))
  using assms
  by (induct B, simp, simp add: image-mset-Diff)

lemma msub-list-subtract-elem-cons-msub:
  assumes mset Ξ ⊆# mset Γ
      and ψ ∈ set (Γ ⊖ Ξ)
    shows mset (ψ # Ξ) ⊆# mset Γ
proof −
  have ∀ Γ. mset Ξ ⊆# mset Γ
           ⟶ ψ ∈ set (Γ ⊖ Ξ) −−> mset (ψ # Ξ) ⊆# mset Γ
  proof(induct Ξ)
    case Nil
    then show ?case by simp
  next
    case (Cons ξ Ξ)
    {
      fix Γ
      assume
        mset (ξ # Ξ) ⊆# mset Γ
        ψ ∈ set (Γ ⊖ (ξ # Ξ))
      hence
         ξ ∈ set Γ
         mset Ξ ⊆# mset (remove1 ξ Γ)
         ψ ∈ set ((remove1 ξ Γ) ⊖ Ξ)
        by (simp,
            metis
              ex-mset
              list.set-intros(1)
              mset.simps(2)
              mset-eq-setD
              subset-mset.le-iff-add
              union-mset-add-mset-left,
            metis
              list-subtract.simps(1)
              list-subtract.simps(2)
              list-subtract-monotonic
              remove-hd,
            simp,
            metis
```

```
        list-subtract-remove1-cons-perm
        perm-set-eq)
    with Cons.hyps have
      mset Γ = mset (ξ # (remove1 ξ Γ))
      mset (ψ # Ξ) ⊆# mset (remove1 ξ Γ)
      by (simp, blast)
    hence mset (ψ # ξ # Ξ) ⊆# mset Γ
      by (simp,
          metis
            add-mset-commute
            mset-subset-eq-add-mset-cancel)
  }
  then show ?case by auto
 qed
 thus ?thesis using assms by blast
qed
```

4.7 Tuple Lists

```
lemma remove1-pairs-list-projections-fst:
  assumes (γ,σ) ∈# mset Φ
  shows mset (map fst (remove1 (γ, σ) Φ)) = mset (map fst Φ) − {# γ #}
using assms
proof (induct Φ)
  case Nil
  then show ?case by simp
next
  case (Cons φ Φ)
  assume (γ, σ) ∈# mset (φ # Φ)
  show ?case
  proof (cases φ = (γ, σ))
    assume φ = (γ, σ)
    then show ?thesis by simp
  next
    assume φ ≠ (γ, σ)
    then have add-mset φ (mset Φ − {#(γ, σ)#})
            = add-mset φ (mset Φ) − {#(γ, σ)#}
        by force
    then have add-mset (fst φ) (image-mset fst (mset Φ − {#(γ, σ)#}))
              = add-mset (fst φ) (image-mset fst (mset Φ)) − {#γ#}
      by (metis (no-types) Cons.prems
                   add-mset-remove-trivial
                   fst-conv
                   image-mset-add-mset
                   insert-DiffM mset.simps(2))
    with ‹φ ≠ (γ, σ)› show ?thesis
      by simp
  qed
qed
```

lemma *remove1-pairs-list-projections-snd*:
  assumes $(\gamma,\sigma) \in\# \; mset\ \Phi$
  shows $mset\ (map\ snd\ (remove1\ (\gamma,\ \sigma)\ \Phi)) = mset\ (map\ snd\ \Phi) - \{\#\ \sigma\ \#\}$
using *assms*
proof (*induct* Φ)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Cons* φ Φ)
  assume $(\gamma,\ \sigma) \in\#\ mset\ (\varphi\ \#\ \Phi)$
  show *?case*
  proof (*cases* $\varphi = (\gamma,\ \sigma)$)
    assume $\varphi = (\gamma,\ \sigma)$
    then show *?thesis* **by** *simp*
  next
    assume $\varphi \neq (\gamma,\ \sigma)$
    then have $add\text{-}mset\ (snd\ \varphi)\ (image\text{-}mset\ snd\ (mset\ \Phi - \{\#(\gamma,\ \sigma)\#\}))$
          $= image\text{-}mset\ snd\ (mset\ (\varphi\ \#\ \Phi) - \{\#(\gamma,\ \sigma)\#\})$
      by *auto*
    moreover have $add\text{-}mset\ (snd\ \varphi)\ (image\text{-}mset\ snd\ (mset\ \Phi))$
            $= add\text{-}mset\ \sigma\ (image\text{-}mset\ snd\ (mset\ (\varphi\ \#\ \Phi) - \{\#(\gamma,\ \sigma)\#\}))$
      by (*metis* (*no-types*)
          Cons.prems
          image-mset-add-mset
          insert-DiffM
          mset.simps(*2*)
          snd-conv)
    ultimately
    have $add\text{-}mset\ (snd\ \varphi)\ (image\text{-}mset\ snd\ (mset\ \Phi - \{\#(\gamma,\ \sigma)\#\}))$
          $= add\text{-}mset\ (snd\ \varphi)\ (image\text{-}mset\ snd\ (mset\ \Phi)) - \{\#\sigma\#\}$
      by *simp*
    with ‹$\varphi \neq (\gamma,\ \sigma)$› **show** *?thesis*
      by *simp*
  qed
qed

lemma *triple-list-exists*:
  assumes $mset\ (map\ snd\ \Psi) \subseteq\#\ mset\ \Sigma$
    and $mset\ \Sigma \subseteq\#\ mset\ (map\ snd\ \Delta)$
   shows $\exists\ \Omega.\ map\ (\lambda\ (\psi,\ \sigma,\ \text{-}).\ (\psi,\ \sigma))\ \Omega = \Psi\ \wedge$
          $mset\ (map\ (\lambda\ (\text{-},\ \sigma,\ \gamma).\ (\gamma,\ \sigma))\ \Omega) \subseteq\#\ mset\ \Delta$
  using *assms*(*1*)
proof (*induct* Ψ)
  case *Nil*
  then show *?case* **by** *fastforce*
next
  case (*Cons* ψ Ψ)
  from *Cons* **obtain** Ω **where** Ω:

```
  map (λ (ψ, σ, -). (ψ, σ)) Ω = Ψ
  mset (map (λ (-, σ, γ). (γ, σ)) Ω) ⊆# mset Δ
  by (metis
        (no-types, lifting)
        diff-subset-eq-self
        list.set-intros(1)
        remove1-pairs-list-projections-snd
        remove-hd
        set-mset-mset
        subset-mset.dual-order.trans
        surjective-pairing)
let ?ΔΩ = map (λ (-, σ, γ). (γ, σ)) Ω
let ?ψ = fst ψ
let ?σ = snd ψ
from Cons.prems have add-mset ?σ (image-mset snd (mset Ψ)) ⊆# mset Σ
  by simp
then have mset Σ − {#?σ#} − image-mset snd (mset Ψ)
            ≠ mset Σ − image-mset snd (mset Ψ)
  by (metis
        (no-types)
        insert-subset-eq-iff
        mset-subset-eq-insertD
        multi-drop-mem-not-eq
        subset-mset.diff-add
        subset-mset-def)
hence ?σ ∈# mset Σ − mset (map snd Ψ)
  using diff-single-trivial by fastforce
have mset (map snd (ψ # Ψ)) ⊆# mset (map snd Δ)
  by (meson
        Cons.prems
        ‹mset Σ ⊆# mset (map snd Δ)›
        subset-mset.dual-order.trans)
then have
  mset (map snd Δ) − mset (map snd (ψ # Ψ)) + ({#} + {#snd ψ#})
    = mset (map snd Δ) + ({#} + {#snd ψ#})
       − add-mset (snd ψ) (mset (map snd Ψ))
  by (metis
        (no-types)
        list.simps(9)
        mset.simps(2)
        mset-subset-eq-multiset-union-diff-commute)
then have
  mset (map snd Δ) − mset (map snd (ψ # Ψ)) + ({#} + {#snd ψ#})
    = mset (map snd Δ) − mset (map snd Ψ)
  by auto
hence ?σ ∈# mset (map snd Δ) − mset (map snd Ψ)
  using add-mset-remove-trivial-eq by fastforce
moreover have snd ∘ (λ (ψ, σ, -). (ψ, σ)) = snd ∘ (λ (-, σ, γ). (γ, σ))
  by auto
```

```
  hence map snd (?ΔΩ) = map snd (map (λ (ψ, σ, -). (ψ, σ)) Ω)
    by fastforce
  hence map snd (?ΔΩ) = map snd Ψ
    using Ω(1) by simp
  ultimately have ?σ ∈# mset (map snd Δ) − mset (map snd ?ΔΩ)
    by simp
  hence ?σ ∈# image-mset snd (mset Δ − mset ?ΔΩ)
    using Ω(2) by (metis image-mset-Diff mset-map)
  hence ?σ ∈ snd ‘ set-mset (mset Δ − mset ?ΔΩ)
    by (metis in-image-mset)
  from this obtain ϱ where ϱ:
    snd ϱ = ?σ ϱ ∈# mset Δ − mset ?ΔΩ
    using imageE by auto
  from this obtain γ where
    (γ, ?σ) = ϱ
    by (metis prod.collapse)
  with ϱ(2) have γ: (γ, ?σ) ∈# mset Δ − mset ?ΔΩ by auto
  let ?Ω = (?ψ, ?σ, γ) # Ω
  have map (λ (ψ, σ, -). (ψ, σ)) ?Ω = ψ # Ψ
    using Ω(1) by simp
  moreover
  have A: (γ, snd ψ) = (case (snd ψ, γ) of (a, c) ⇒ (c, a))
    by auto
  have B: mset (map (λ(b, a, c). (c, a)) Ω)
             + {# case (snd ψ, γ) of (a, c) ⇒ (c, a) #}
           = mset (map (λ(b, a, c). (c, a)) ((fst ψ, snd ψ, γ) # Ω))
    by simp
  obtain mm
     :: ('c × 'a) multiset
     ⇒ ('c × 'a) multiset
     ⇒ ('c × 'a) multiset
    where ∀ x0 x1. (∃ v2. x0 = x1 + v2) = (x0 = x1 + mm x0 x1)
    by moura
  then have mset Δ = mset (map (λ(b, a, c). (c, a)) Ω)
                     + mm (mset Δ) (mset (map (λ(b, a, c). (c, a)) Ω))
    by (metis Ω(2) subset-mset.le-iff-add)
  then have mset (map (λ (-, σ, γ). (γ, σ)) ?Ω) ⊆# mset Δ
    using A B by
      (metis
         γ
         add-diff-cancel-left'
         single-subset-iff
         subset-mset.add-le-cancel-left)
  ultimately show ?case by meson
qed
```

4.8 List Intersection

primrec *list-intersect* :: *'a list => 'a list => 'a list* (**infixl** ‹∩› *60*)

```
  where
    - ∩ [] = []
  | xs ∩ (y # ys) =
        (if (y ∈ set xs)
         then (y # (remove1 y xs ∩ ys))
         else (xs ∩ ys))

lemma list-intersect-mset-homomorphism [simp]:
  mset (Φ ∩ Ψ) = mset Φ ∩# mset Ψ
proof −
  have ∀ Φ. mset (Φ ∩ Ψ) = mset Φ ∩# mset Ψ
  proof (induct Ψ)
    case Nil
    then show ?case by simp
  next
    case (Cons ψ Ψ)
    {
      fix Φ
      have mset (Φ ∩ ψ # Ψ) = mset Φ ∩# mset (ψ # Ψ)
        using Cons.hyps
        by (cases ψ ∈ set Φ,
            simp add: inter-add-right2,
            simp add: inter-add-right1)
    }
    then show ?case by blast
  qed
  thus ?thesis by simp
qed

lemma list-intersect-left-empty [simp]: [] ∩ Φ = [] by (induct Φ, simp+)

lemma list-diff-intersect-comp:
  mset Φ = mset (Φ ⊖ Ψ) + mset (Φ ∩ Ψ)
  by (metis
        diff-intersect-left-idem
        list-intersect-mset-homomorphism
        list-subtract-mset-homomorphism
        subset-mset.inf-le1
        subset-mset.le-imp-diff-is-add)

lemma list-intersect-left-project: mset (Φ ∩ Ψ) ⊆# mset Φ
  by simp

lemma list-intersect-right-project: mset (Φ ∩ Ψ) ⊆# mset Ψ
  by simp

end
```

Chapter 5

Classical Logic Connectives

theory *Classical-Connectives*
  imports
    Classical-Logic-Completeness
    List-Utilities
begin

Here we define the usual connectives for classical logic.

unbundle *no funcset-syntax*

5.1 Verum

definition (**in** *classical-logic*) *verum* :: $'a$ (‹⊤›)
  where
    $\top = \bot \to \bot$

lemma (**in** *classical-logic*) *verum-tautology* [*simp*]: $\vdash \top$
  by (*metis list-implication.simps*(*1*) *list-implication-axiom-k verum-def*)

lemma *verum-semantics* [*simp*]:
  $\mathfrak{M} \models_{prop} \top$
  unfolding *verum-def* **by** *simp*

lemma (**in** *classical-logic*) *verum-embedding* [*simp*]:
  $(\!| \top |\!) = \top$
  by (*simp add*: *classical-logic-class.verum-def verum-def*)

5.2 Conjunction

definition (**in** *classical-logic*)
  conjunction :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⊓› *67*)
  where
    $\varphi \sqcap \psi = (\varphi \to \psi \to \bot) \to \bot$

primrec (**in** *classical-logic*)
arbitrary-conjunction :: *'a list* ⇒ *'a* (‹$\bigsqcap$›)
where
$\bigsqcap$ [] = ⊤
| $\bigsqcap$ (φ # Φ) = φ ⊓ $\bigsqcap$ Φ

lemma (**in** *classical-logic*) *conjunction-introduction*:
⊢ φ → ψ → (φ ⊓ ψ)
by (*metis*
modus-ponens
conjunction-def
list-flip-implication1
list-implication.simps(*1*)
list-implication.simps(*2*))

lemma (**in** *classical-logic*) *conjunction-left-elimination*:
⊢ (φ ⊓ ψ) → φ
by (*metis* (*full-types*)
Peirces-law
pseudo-scotus
conjunction-def
list-deduction-base-theory
list-deduction-modus-ponens
list-deduction-theorem
list-deduction-weaken)

lemma (**in** *classical-logic*) *conjunction-right-elimination*:
⊢ (φ ⊓ ψ) → ψ
by (*metis* (*full-types*)
axiom-k
Contraposition
modus-ponens
conjunction-def
flip-hypothetical-syllogism
flip-implication)

lemma (**in** *classical-logic*) *conjunction-embedding* [*simp*]:
⦇ φ ⊓ ψ ⦈ = ⦇ φ ⦈ ⊓ ⦇ ψ ⦈
unfolding *conjunction-def classical-logic-class.conjunction-def*
by *simp*

lemma *conjunction-semantics* [*simp*]:
$\mathfrak{M} \models_{prop} \varphi \sqcap \psi = (\mathfrak{M} \models_{prop} \varphi \wedge \mathfrak{M} \models_{prop} \psi)$
unfolding *conjunction-def* **by** *simp*

5.3 Biconditional

definition (**in** *classical-logic*) *biconditional* :: *'a* ⇒ *'a* ⇒ *'a* (**infixr** ‹↔› *75*)
where

$\varphi \leftrightarrow \psi = (\varphi \to \psi) \sqcap (\psi \to \varphi)$

lemma (**in** *classical-logic*) *biconditional-introduction*:
$\vdash (\varphi \to \psi) \to (\psi \to \varphi) \to (\varphi \leftrightarrow \psi)$
by (*simp add*: *biconditional-def conjunction-introduction*)

lemma (**in** *classical-logic*) *biconditional-left-elimination*:
$\vdash (\varphi \leftrightarrow \psi) \to \varphi \to \psi$
by (*simp add*: *biconditional-def conjunction-left-elimination*)

lemma (**in** *classical-logic*) *biconditional-right-elimination*:
$\vdash (\varphi \leftrightarrow \psi) \to \psi \to \varphi$
by (*simp add*: *biconditional-def conjunction-right-elimination*)

lemma (**in** *classical-logic*) *biconditional-embedding* [*simp*]:
$(\!|\ \varphi \leftrightarrow \psi\ |\!) = (\!|\ \varphi\ |\!) \leftrightarrow (\!|\ \psi\ |\!)$
unfolding *biconditional-def classical-logic-class.biconditional-def*
by *simp*

lemma *biconditional-semantics* [*simp*]:
$\mathfrak{M} \models_{prop} \varphi \leftrightarrow \psi = (\mathfrak{M} \models_{prop} \varphi \longleftrightarrow \mathfrak{M} \models_{prop} \psi)$
unfolding *biconditional-def*
by (*simp*, *blast*)

5.4 Negation

definition (**in** *classical-logic*) *negation* :: *'a* ⇒ *'a* (‹∼›)
where
$\sim \varphi = \varphi \to \bot$

lemma (**in** *classical-logic*) *negation-introduction*:
$\vdash (\varphi \to \bot) \to \sim \varphi$
unfolding *negation-def*
by (*metis axiom-k modus-ponens implication-absorption*)

lemma (**in** *classical-logic*) *negation-elimination*:
$\vdash \sim \varphi \to (\varphi \to \bot)$
unfolding *negation-def*
by (*metis axiom-k modus-ponens implication-absorption*)

lemma (**in** *classical-logic*) *negation-embedding* [*simp*]:
$(\!|\ \sim \varphi\ |\!) = \sim (\!|\ \varphi\ |\!)$
by (*simp add*:
classical-logic-class.negation-def
negation-def)

lemma *negation-semantics* [*simp*]:
$\mathfrak{M} \models_{prop} \sim \varphi = (\neg\ \mathfrak{M} \models_{prop} \varphi)$
unfolding *negation-def*

by *simp*

5.5 Disjunction

definition (**in** *classical-logic*) *disjunction* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹⊔› *67*)
where
$\varphi \sqcup \psi = (\varphi \to \bot) \to \psi$

primrec (**in** *classical-logic*) *arbitrary-disjunction* :: $'a$ *list* $\Rightarrow 'a$ (‹⨆›)
where
$\bigsqcup [] = \bot$
| $\bigsqcup (\varphi \# \Phi) = \varphi \sqcup \bigsqcup \Phi$

lemma (**in** *classical-logic*) *disjunction-elimination*:
$\vdash (\varphi \to \chi) \to (\psi \to \chi) \to (\varphi \sqcup \psi) \to \chi$
proof −
let $?\Gamma = [\varphi \to \chi, \psi \to \chi, \varphi \sqcup \psi]$
have $?\Gamma :\vdash (\varphi \to \bot) \to \chi$
unfolding *disjunction-def*
by (*metis hypothetical-syllogism*
list-deduction-def
list-implication.simps(*1*)
list-implication.simps(*2*)
set-deduction-base-theory
set-deduction-theorem
set-deduction-weaken)
hence $?\Gamma :\vdash \chi$
using *excluded-middle-elimination*
list-deduction-modus-ponens
list-deduction-theorem
list-deduction-weaken
by *blast*
thus *?thesis*
unfolding *list-deduction-def*
by *simp*
qed

lemma (**in** *classical-logic*) *disjunction-left-introduction*:
$\vdash \varphi \to (\varphi \sqcup \psi)$
unfolding *disjunction-def*
by (*metis modus-ponens*
pseudo-scotus
flip-implication)

lemma (**in** *classical-logic*) *disjunction-right-introduction*:
$\vdash \psi \to (\varphi \sqcup \psi)$
unfolding *disjunction-def*
using *axiom-k*
by *simp*

lemma (**in** *classical-logic*) *disjunction-embedding* [*simp*]:
$⦇ \varphi \sqcup \psi ⦈ = ⦇ \varphi ⦈ \sqcup ⦇ \psi ⦈$
unfolding *disjunction-def classical-logic-class.disjunction-def*
by *simp*

lemma *disjunction-semantics* [*simp*]:
$\mathfrak{M} \models_{prop} \varphi \sqcup \psi = (\mathfrak{M} \models_{prop} \varphi \vee \mathfrak{M} \models_{prop} \psi)$
unfolding *disjunction-def*
by (*simp*, *blast*)

5.6 Mutual Exclusion

primrec (**in** *classical-logic*) *exclusive* :: *'a list* ⇒ *'a* (‹∐›)
where
$\coprod \; [] = \top$
$| \coprod \; (\varphi \# \Phi) = \sim (\varphi \sqcap \bigsqcup \Phi) \sqcap \coprod \Phi$

5.7 Subtraction

definition (**in** *classical-logic*) *subtraction* :: *'a* ⇒ *'a* ⇒ *'a* (**infixl** ‹\› *69*)
where $\varphi \setminus \psi = \varphi \sqcap \sim \psi$

lemma (**in** *classical-logic*) *subtraction-embedding* [*simp*]:
$⦇ \varphi \setminus \psi ⦈ = ⦇ \varphi ⦈ \setminus ⦇ \psi ⦈$
unfolding *subtraction-def classical-logic-class.subtraction-def*
by *simp*

5.8 Negated Lists

definition (**in** *classical-logic*) *map-negation* :: *'a list* ⇒ *'a list* (‹**∼**›)
where [*simp*]: **∼** $\Phi \equiv$ *map* $\sim \Phi$

5.9 Common (& Uncommon) Identities

5.9.1 Biconditional Equivalence Relation

lemma (**in** *classical-logic*) *biconditional-reflection*:
$\vdash \varphi \leftrightarrow \varphi$
by (*meson*
axiom-k
modus-ponens
biconditional-introduction
implication-absorption)

lemma (**in** *classical-logic*) *biconditional-symmetry*:
$\vdash (\varphi \leftrightarrow \psi) \leftrightarrow (\psi \leftrightarrow \varphi)$

by (*metis* (*full-types*) *modus-ponens*
biconditional-def
conjunction-def
flip-hypothetical-syllogism
flip-implication)

lemma (**in** *classical-logic*) *biconditional-symmetry-rule*:
$\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \psi \leftrightarrow \varphi$
by (*meson modus-ponens*
biconditional-introduction
biconditional-left-elimination
biconditional-right-elimination)

lemma (**in** *classical-logic*) *biconditional-transitivity*:
$\vdash (\varphi \leftrightarrow \psi) \rightarrow (\psi \leftrightarrow \chi) \rightarrow (\varphi \leftrightarrow \chi)$
proof −
have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} ((\langle\varphi\rangle \leftrightarrow \langle\psi\rangle) \rightarrow ((\langle\psi\rangle \leftrightarrow \langle\chi\rangle) \rightarrow ((\langle\varphi\rangle \leftrightarrow \langle\chi\rangle))$
by *simp*
hence $\vdash (\!|\ ((\langle\varphi\rangle \leftrightarrow \langle\psi\rangle) \rightarrow ((\langle\psi\rangle \leftrightarrow \langle\chi\rangle) \rightarrow ((\langle\varphi\rangle \leftrightarrow \langle\chi\rangle)\ |\!)$
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *biconditional-transitivity-rule*:
$\vdash \varphi \leftrightarrow \psi \Longrightarrow \vdash \psi \leftrightarrow \chi \Longrightarrow \vdash \varphi \leftrightarrow \chi$
using *modus-ponens biconditional-transitivity* **by** *blast*

5.9.2 Biconditional Weakening

lemma (**in** *classical-logic*) *biconditional-weaken*:
assumes $\Gamma \Vdash \varphi \leftrightarrow \psi$
shows $\Gamma \Vdash \varphi = \Gamma \Vdash \psi$
by (*metis assms*
biconditional-left-elimination
biconditional-right-elimination
set-deduction-modus-ponens
set-deduction-weaken)

lemma (**in** *classical-logic*) *list-biconditional-weaken*:
assumes $\Gamma :\vdash \varphi \leftrightarrow \psi$
shows $\Gamma :\vdash \varphi = \Gamma :\vdash \psi$
by (*metis assms*
biconditional-left-elimination
biconditional-right-elimination
list-deduction-modus-ponens
list-deduction-weaken)

lemma (**in** *classical-logic*) *weak-biconditional-weaken*:
assumes $\vdash \varphi \leftrightarrow \psi$

shows $\vdash \varphi = \vdash \psi$
by (*metis assms*
biconditional-left-elimination
biconditional-right-elimination
modus-ponens)

5.9.3 Conjunction Identities

lemma (**in** *classical-logic*) *conjunction-negation-identity*:
$\vdash \sim (\varphi \sqcap \psi) \leftrightarrow (\varphi \rightarrow \psi \rightarrow \bot)$
by (*metis Contraposition*
double-negation-converse
modus-ponens
biconditional-introduction
conjunction-def
negation-def)

lemma (**in** *classical-logic*) *conjunction-set-deduction-equivalence* [*simp*]:
$\Gamma \Vdash \varphi \sqcap \psi = (\Gamma \Vdash \varphi \wedge \Gamma \Vdash \psi)$
by (*metis set-deduction-weaken* [**where** Γ=Γ]
set-deduction-modus-ponens [**where** Γ=Γ]
conjunction-introduction
conjunction-left-elimination
conjunction-right-elimination)

lemma (**in** *classical-logic*) *conjunction-list-deduction-equivalence* [*simp*]:
$\Gamma :\vdash \varphi \sqcap \psi = (\Gamma :\vdash \varphi \wedge \Gamma :\vdash \psi)$
by (*metis list-deduction-weaken* [**where** Γ=Γ]
list-deduction-modus-ponens [**where** Γ=Γ]
conjunction-introduction
conjunction-left-elimination
conjunction-right-elimination)

lemma (**in** *classical-logic*) *weak-conjunction-deduction-equivalence* [*simp*]:
$\vdash \varphi \sqcap \psi = (\vdash \varphi \wedge \vdash \psi)$
by (*metis conjunction-set-deduction-equivalence set-deduction-base-theory*)

lemma (**in** *classical-logic*) *conjunction-set-deduction-arbitrary-equivalence* [*simp*]:
$\Gamma \Vdash \prod \Phi = (\forall \varphi \in set\ \Phi.\ \Gamma \Vdash \varphi)$
by (*induct* Φ, *simp add*: *set-deduction-weaken*, *simp*)

lemma (**in** *classical-logic*) *conjunction-list-deduction-arbitrary-equivalence* [*simp*]:
$\Gamma :\vdash \prod \Phi = (\forall \varphi \in set\ \Phi.\ \Gamma :\vdash \varphi)$
by (*induct* Φ, *simp add*: *list-deduction-weaken*, *simp*)

lemma (**in** *classical-logic*) *weak-conjunction-deduction-arbitrary-equivalence* [*simp*]:
$\vdash \prod \Phi = (\forall \varphi \in set\ \Phi.\ \vdash \varphi)$
by (*induct* Φ, *simp+*)

lemma (**in** *classical-logic*) *conjunction-commutativity*:
  $\vdash (\psi \sqcap \varphi) \leftrightarrow (\varphi \sqcap \psi)$
  by (*metis*
      (*full-types*)
      modus-ponens
      biconditional-introduction
      conjunction-def
      flip-hypothetical-syllogism
      flip-implication)

lemma (**in** *classical-logic*) *conjunction-associativity*:
  $\vdash ((\varphi \sqcap \psi) \sqcap \chi) \leftrightarrow (\varphi \sqcap (\psi \sqcap \chi))$
proof −
  have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} (((\langle\varphi\rangle \sqcap \langle\psi\rangle) \sqcap \langle\chi\rangle) \leftrightarrow (\langle\varphi\rangle \sqcap (\langle\psi\rangle \sqcap \langle\chi\rangle)))$
    by *simp*
  hence $\vdash (\!|\ (((\langle\varphi\rangle \sqcap \langle\psi\rangle) \sqcap \langle\chi\rangle) \leftrightarrow (\langle\varphi\rangle \sqcap (\langle\psi\rangle \sqcap \langle\chi\rangle)))\ |\!)$
    using *propositional-semantics* **by** *blast*
  thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *arbitrary-conjunction-antitone*:
  set $\Phi \subseteq$ *set* $\Psi \Longrightarrow\ \vdash \prod\ \Psi \rightarrow \prod\ \Phi$
proof −
  have $\forall\ \Phi.$ *set* $\Phi \subseteq$ *set* $\Psi \longrightarrow\ \vdash \prod\ \Psi \rightarrow \prod\ \Phi$
  proof (*induct* Ψ)
    case *Nil*
    then show *?case*
      by (*simp add*: *pseudo-scotus verum-def*)
  next
    case (*Cons* ψ Ψ)
    {
      fix Φ
      assume *set* $\Phi \subseteq$ *set* $(\psi\ \#\ \Psi)$
      have $\vdash \prod\ (\psi\ \#\ \Psi) \rightarrow \prod\ \Phi$
      proof (*cases* $\psi \in$ *set* Φ)
        assume $\psi \in$ *set* Φ
            have $\forall\ \varphi \in$ *set* $\Phi.\ \vdash \prod\ \Phi \leftrightarrow (\varphi \sqcap \prod\ ($*removeAll* $\varphi\ \Phi))$
        proof (*induct* Φ)
          case *Nil*
          then show *?case* **by** *simp*
        next
          case (*Cons* χ Φ)
          {
            fix φ
            assume $\varphi \in$ *set* $(\chi\ \#\ \Phi)$
            have $\vdash \prod\ (\chi\ \#\ \Phi) \leftrightarrow (\varphi \sqcap \prod\ ($*removeAll* $\varphi\ (\chi\ \#\ \Phi)))$
            proof *cases*
              assume $\varphi \in$ *set* Φ
              hence $\vdash \prod\ \Phi \leftrightarrow (\varphi \sqcap \prod\ ($*removeAll* $\varphi\ \Phi))$

using *Cons.hyps* ‹$\varphi \in$ *set* Φ›
by *auto*
moreover
have $\vdash (\bigsqcap\ \Phi \leftrightarrow (\varphi \sqcap \bigsqcap\ (\mathit{removeAll}\ \varphi\ \Phi))) \rightarrow$
$(\chi \sqcap \bigsqcap\ \Phi) \leftrightarrow (\varphi \sqcap \chi \sqcap \bigsqcap\ (\mathit{removeAll}\ \varphi\ \Phi))$
proof −
have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop}$
$(\langle\bigsqcap\ \Phi\rangle \leftrightarrow (\langle\varphi\rangle \sqcap \langle\bigsqcap\ (\mathit{removeAll}\ \varphi\ \Phi)\rangle))$
$\rightarrow (\langle\chi\rangle \sqcap \langle\bigsqcap\ \Phi\rangle)$
$\leftrightarrow (\langle\varphi\rangle \sqcap \langle\chi\rangle \sqcap \langle\bigsqcap\ (\mathit{removeAll}\ \varphi\ \Phi)\rangle)$
by *auto*
hence $\vdash (\!|\ (\langle\bigsqcap\ \Phi\rangle \leftrightarrow (\langle\varphi\rangle \sqcap \langle\bigsqcap\ (\mathit{removeAll}\ \varphi\ \Phi)\rangle))$
$\rightarrow (\langle\chi\rangle \sqcap \langle\bigsqcap\ \Phi\rangle)$
$\leftrightarrow (\langle\varphi\rangle \sqcap \langle\chi\rangle \sqcap \langle\bigsqcap\ (\mathit{removeAll}\ \varphi\ \Phi)\rangle)\ |\!)$
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
ultimately have $\vdash \bigsqcap\ (\chi\ \#\ \Phi) \leftrightarrow (\varphi \sqcap \chi \sqcap \bigsqcap\ (\mathit{removeAll}\ \varphi\ \Phi))$
using *modus-ponens* **by** *auto*
show *?thesis*
proof *cases*
assume $\varphi = \chi$
moreover
{
fix φ
have $\vdash (\chi \sqcap \varphi) \rightarrow (\chi \sqcap \chi \sqcap \varphi)$
unfolding *conjunction-def*
by (*meson*
axiom-s
double-negation
modus-ponens
flip-hypothetical-syllogism
flip-implication)
} **note** *tautology* = *this*
from ‹$\vdash \bigsqcap\ (\chi\ \#\ \Phi) \leftrightarrow (\varphi \sqcap \chi \sqcap \bigsqcap\ (\mathit{removeAll}\ \varphi\ \Phi))$›
‹$\varphi = \chi$›
have $\vdash (\chi \sqcap \bigsqcap\ (\mathit{removeAll}\ \chi\ \Phi)) \rightarrow (\chi \sqcap \bigsqcap\ \Phi)$
unfolding *biconditional-def*
by (*simp*, *metis tautology hypothetical-syllogism modus-ponens*)
moreover
from ‹$\vdash \bigsqcap\ (\chi\ \#\ \Phi) \leftrightarrow (\varphi \sqcap \chi \sqcap \bigsqcap\ (\mathit{removeAll}\ \varphi\ \Phi))$›
‹$\varphi = \chi$›
have $\vdash (\chi \sqcap \bigsqcap\ \Phi) \rightarrow (\chi \sqcap \bigsqcap\ (\mathit{removeAll}\ \chi\ \Phi))$
unfolding *biconditional-def*
by (*simp*,
metis conjunction-right-elimination
hypothetical-syllogism
modus-ponens)
ultimately show *?thesis*

```
          unfolding biconditional-def
          by simp
      next
        assume φ ≠ χ
        then show ?thesis
          using ‹⊢ ⨅ (χ # Φ) ↔ (φ ⊓ χ ⊓ ⨅ (removeAll φ Φ))›
          by simp
      qed
    next
      assume φ ∉ set Φ
      hence φ = χ χ ∉ set Φ
        using ‹φ ∈ set (χ # Φ)› by auto
      then show ?thesis
        using biconditional-reflection
        by simp
    qed
  }
  thus ?case by blast
qed
hence ⊢ (ψ ⊓ ⨅ (removeAll ψ Φ)) → ⨅ Φ
  using modus-ponens biconditional-right-elimination ‹ψ ∈ set Φ›
  by blast
moreover
from ‹ψ ∈ set Φ› ‹set Φ ⊆ set (ψ # Ψ)› Cons.hyps
have ⊢ ⨅ Ψ → ⨅ (removeAll ψ Φ)
  by (simp add: subset-insert-iff insert-absorb)
hence ⊢ (ψ ⊓ ⨅ Ψ) → (ψ ⊓ ⨅ (removeAll ψ Φ))
  unfolding conjunction-def
  using
    modus-ponens
    hypothetical-syllogism
    flip-hypothetical-syllogism
  by meson
ultimately have ⊢ (ψ ⊓ ⨅ Ψ) → ⨅ Φ
  using modus-ponens hypothetical-syllogism
  by blast
thus ?thesis
  by simp
next
  assume ψ ∉ set Φ
  hence ⊢ ⨅ Ψ → ⨅ Φ
    using Cons.hyps ‹set Φ ⊆ set (ψ # Ψ)›
    by auto
  hence ⊢ (ψ ⊓ ⨅ Ψ) → ⨅ Φ
    unfolding conjunction-def
    by (metis
          modus-ponens
          conjunction-def
          conjunction-right-elimination
```

hypothetical-syllogism)
thus *?thesis*
by *simp*
qed
}
thus *?case* **by** *blast*
qed
thus *set* $\Phi \subseteq$ *set* $\Psi \Longrightarrow \vdash \bigsqcap \Psi \to \bigsqcap \Phi$ **by** *blast*
qed

lemma (**in** *classical-logic*) *arbitrary-conjunction-remdups*:
$\vdash (\bigsqcap \Phi) \leftrightarrow \bigsqcap$ (*remdups* Φ)
by (*simp add*: *arbitrary-conjunction-antitone biconditional-def*)

lemma (**in** *classical-logic*) *curry-uncurry*:
$\vdash (\varphi \to \psi \to \chi) \leftrightarrow ((\varphi \sqcap \psi) \to \chi)$
proof −
have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} (\langle\varphi\rangle \to \langle\psi\rangle \to \langle\chi\rangle) \leftrightarrow ((\langle\varphi\rangle \sqcap \langle\psi\rangle) \to \langle\chi\rangle)$
by *auto*
hence $\vdash (\!|\ (\langle\varphi\rangle \to \langle\psi\rangle \to \langle\chi\rangle) \leftrightarrow ((\langle\varphi\rangle \sqcap \langle\psi\rangle) \to \langle\chi\rangle)\ |\!)$
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *list-curry-uncurry*:
$\vdash (\Phi :\to \chi) \leftrightarrow (\bigsqcap \Phi \to \chi)$
proof (*induct* Φ)
case *Nil*
have $\vdash \chi \leftrightarrow (\top \to \chi)$
unfolding *biconditional-def*
conjunction-def
verum-def
using
axiom-k
ex-falso-quodlibet
modus-ponens
conjunction-def
excluded-middle-elimination
set-deduction-base-theory
conjunction-set-deduction-equivalence
by *metis*
with *Nil* **show** *?case*
by *simp*
next
case (*Cons* φ Φ)
have $\vdash ((\varphi \# \Phi) :\to \chi) \leftrightarrow (\varphi \to (\Phi :\to \chi))$
by (*simp add*: *biconditional-reflection*)
with *Cons* **have** $\vdash ((\varphi \# \Phi) :\to \chi) \leftrightarrow (\varphi \to \bigsqcap \Phi \to \chi)$
by (*metis modus-ponens*

biconditional-def
hypothetical-syllogism
list-implication.simps(*2*)
weak-conjunction-deduction-equivalence)
with *curry-uncurry* [**where** $?\varphi{=}\varphi$ **and** $?\psi{=}\bigsqcap\ \Phi$ **and** $?\chi{=}\chi$]
show *?case*
unfolding *biconditional-def*
by (*simp*, *metis modus-ponens hypothetical-syllogism*)
qed

5.9.4 Disjunction Identities

lemma (**in** *classical-logic*) *bivalence*:
$\vdash \sim \varphi \sqcup \varphi$
by (*simp add*: *double-negation disjunction-def negation-def*)

lemma (**in** *classical-logic*) *implication-equivalence*:
$\vdash (\sim \varphi \sqcup \psi) \leftrightarrow (\varphi \rightarrow \psi)$
by (*metis double-negation-converse*
modus-ponens
biconditional-introduction
bivalence
disjunction-def
flip-hypothetical-syllogism
negation-def)

lemma (**in** *classical-logic*) *disjunction-commutativity*:
$\vdash (\psi \sqcup \varphi) \leftrightarrow (\varphi \sqcup \psi)$
by (*meson modus-ponens*
biconditional-introduction
disjunction-elimination
disjunction-left-introduction
disjunction-right-introduction)

lemma (**in** *classical-logic*) *disjunction-associativity*:
$\vdash ((\varphi \sqcup \psi) \sqcup \chi) \leftrightarrow (\varphi \sqcup (\psi \sqcup \chi))$
proof −
have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} (((\langle\varphi\rangle \sqcup \langle\psi\rangle) \sqcup \langle\chi\rangle) \leftrightarrow (\langle\varphi\rangle \sqcup (\langle\psi\rangle \sqcup \langle\chi\rangle))$
by *simp*
hence $\vdash (\!|\ (((\langle\varphi\rangle \sqcup \langle\psi\rangle) \sqcup \langle\chi\rangle) \leftrightarrow (\langle\varphi\rangle \sqcup (\langle\psi\rangle \sqcup \langle\chi\rangle))\ |\!)$
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *arbitrary-disjunction-monotone*:
set $\Phi \subseteq$ *set* $\Psi \Longrightarrow\ \vdash \bigsqcup\ \Phi \rightarrow \bigsqcup\ \Psi$
proof −
have $\forall\ \Phi.$ *set* $\Phi \subseteq$ *set* $\Psi \longrightarrow\ \vdash \bigsqcup\ \Phi \rightarrow \bigsqcup\ \Psi$
proof (*induct* Ψ)

```
  case Nil
  then show ?case using verum-def verum-tautology by auto
next
  case (Cons ψ Ψ)
  {
    fix Φ
    assume set Φ ⊆ set (ψ # Ψ)
    have ⊢ ⨆ Φ → ⨆ (ψ # Ψ)
    proof cases
      assume ψ ∈ set Φ
      have ∀ φ ∈ set Φ. ⊢ ⨆ Φ ↔ (φ ⊔ ⨆ (removeAll φ Φ))
      proof (induct Φ)
        case Nil
        then show ?case by simp
      next
        case (Cons χ Φ)
        {
          fix φ
          assume φ ∈ set (χ # Φ)
          have ⊢ ⨆ (χ # Φ) ↔ (φ ⊔ ⨆ (removeAll φ (χ # Φ)))
          proof cases
            assume φ ∈ set Φ
            hence ⊢ ⨆ Φ ↔ (φ ⊔ ⨆ (removeAll φ Φ))
              using Cons.hyps ‹φ ∈ set Φ›
              by auto
            moreover
            have ⊢ (⨆ Φ ↔ (φ ⊔ ⨆ (removeAll φ Φ))) →
                    (χ ⊔ ⨆ Φ) ↔ (φ ⊔ χ ⊔ ⨆ (removeAll φ Φ))
            proof −
              have ∀ 𝔐. 𝔐 ⊨_prop
                        ((⟨⨆ Φ⟩ ↔ (⟨φ⟩ ⊔ ⟨⨆ (removeAll φ Φ)⟩))
                         → (⟨χ⟩ ⊔ ⟨⨆ Φ⟩)
                              ↔ (⟨φ⟩ ⊔ ⟨χ⟩ ⊔ ⟨⨆ (removeAll φ Φ)⟩))
                by auto
              hence ⊢ ⦇ ((⟨⨆ Φ⟩ ↔ (⟨φ⟩ ⊔ ⟨⨆ (removeAll φ Φ)⟩))
                         → (⟨χ⟩ ⊔ ⟨⨆ Φ⟩)
                              ↔ (⟨φ⟩ ⊔ ⟨χ⟩ ⊔ ⟨⨆ (removeAll φ Φ)⟩)) ⦈
                using propositional-semantics by blast
              thus ?thesis by simp
            qed
            ultimately have ⊢ ⨆ (χ # Φ) ↔ (φ ⊔ χ ⊔ ⨆ (removeAll φ Φ))
              using modus-ponens by auto
            show ?thesis
            proof cases
              assume φ = χ
              then show ?thesis
                using ‹⊢ ⨆ (χ # Φ) ↔ (φ ⊔ χ ⊔ ⨆ (removeAll φ Φ))›
                unfolding biconditional-def
                by (simp add: disjunction-def,
```

```
              meson
                axiom-k
                modus-ponens
                flip-hypothetical-syllogism
                implication-absorption)
        next
          assume φ ≠ χ
          then show ?thesis
            using ‹⊢ ⨆ (χ # Φ) ↔ (φ ⊔ χ ⊔ ⨆ (removeAll φ Φ))›
            by simp
        qed
      next
        assume φ ∉ set Φ
        hence φ = χ χ ∉ set Φ
          using ‹φ ∈ set (χ # Φ)› by auto
        then show ?thesis
          using biconditional-reflection
          by simp
      qed
    }
    thus ?case by blast
  qed
  hence ⊢ ⨆ Φ → (ψ ⊔ ⨆ (removeAll ψ Φ))
    using modus-ponens biconditional-left-elimination ‹ψ ∈ set Φ›
    by blast
  moreover
  from ‹ψ ∈ set Φ› ‹set Φ ⊆ set (ψ # Ψ)› Cons.hyps
  have ⊢ ⨆ (removeAll ψ Φ) → ⨆ Ψ
    by (simp add: subset-insert-iff insert-absorb)
  hence ⊢ (ψ ⊔ ⨆ (removeAll ψ Φ)) → ⨆ (ψ # Ψ)
    using
      modus-ponens
      disjunction-def
      hypothetical-syllogism
    by fastforce
  ultimately show ?thesis
    by (simp, metis modus-ponens hypothetical-syllogism)
next
  assume ψ ∉ set Φ
  hence ⊢ ⨆ Φ → ⨆ Ψ
    using Cons.hyps ‹set Φ ⊆ set (ψ # Ψ)›
    by auto
  then show ?thesis
    by (metis
          arbitrary-disjunction.simps(2)
          disjunction-def
          list-deduction-def
          list-deduction-theorem
          list-deduction-weaken
```

```
          list-implication.simps(1)
          list-implication.simps(2))
    qed
  }
  then show ?case by blast
 qed
 thus set Φ ⊆ set Ψ ⟹ ⊢ ⨆ Φ → ⨆ Ψ by blast
qed

lemma (in classical-logic) arbitrary-disjunction-remdups:
 ⊢ (⨆ Φ) ↔ ⨆ (remdups Φ)
 by (simp add: arbitrary-disjunction-monotone biconditional-def)

lemma (in classical-logic) arbitrary-disjunction-exclusion-MCS:
 assumes MCS Ω
 shows ⨆ Ψ ∉ Ω ≡ ∀ ψ ∈ set Ψ. ψ ∉ Ω
proof (induct Ψ)
 case Nil
 then show ?case
  using
   assms
   formula-consistent-def
   formula-maximally-consistent-set-def-def
   maximally-consistent-set-def
   set-deduction-reflection
  by (simp, blast)
next
 case (Cons ψ Ψ)
 have ⨆ (ψ # Ψ) ∉ Ω = (ψ ∉ Ω ∧ ⨆ Ψ ∉ Ω)
  by (simp add: disjunction-def,
    meson
     assms
     formula-consistent-def
     formula-maximally-consistent-set-def-def
     formula-maximally-consistent-set-def-implication
     maximally-consistent-set-def
     set-deduction-reflection)
 thus ?case using Cons.hyps by simp
qed

lemma (in classical-logic) contra-list-curry-uncurry:
 ⊢ (Φ :→ χ) ↔ (∼ χ → ⨆ (∼ Φ))
proof (induct Φ)
 case Nil
 then show ?case
  by (simp,
     metis
      biconditional-introduction
      bivalence
```

```
          disjunction-def
          double-negation-converse
          modus-ponens
          negation-def)
next
  case (Cons φ Φ)
  hence ⊢ (⨅ Φ → χ) ↔ (∼ χ → ⨆ (∼ Φ))
    by (metis
         biconditional-symmetry-rule
         biconditional-transitivity-rule
         list-curry-uncurry)
  have ⊢ (⨅ (φ # Φ) → χ) ↔ (∼ χ → ⨆ (∼ (φ # Φ)))
  proof −
    have ⊢ (⨅ Φ → χ) ↔ (∼ χ → ⨆ (∼ Φ))
           → ((φ ⊓ ⨅ Φ) → χ) ↔ (∼ χ → (∼ φ ⊔ ⨆ (∼ Φ)))
    proof −
      have
      ∀ 𝔐. 𝔐 ⊨prop
        ((⟨⨅ Φ⟩ → ⟨χ⟩) ↔ (∼ ⟨χ⟩ → ⟨⨆ (∼ Φ)⟩)
           → (((⟨φ⟩ ⊓ ⟨⨅ Φ⟩) → ⟨χ⟩) ↔ (∼ ⟨χ⟩ → (∼ ⟨φ⟩ ⊔ ⟨⨆ (∼ Φ)⟩)))
        by auto
      hence
        ⊢ ⦇ ((⟨⨅ Φ⟩ → ⟨χ⟩) ↔ (∼ ⟨χ⟩ → ⟨⨆ (∼ Φ)⟩)
           → (((⟨φ⟩ ⊓ ⟨⨅ Φ⟩) → ⟨χ⟩) ↔ (∼ ⟨χ⟩ → (∼ ⟨φ⟩ ⊔ ⟨⨆ (∼ Φ)⟩))) ⦈
        using propositional-semantics by blast
      thus ?thesis by simp
    qed
    thus ?thesis
      using ‹⊢ (⨅ Φ → χ) ↔ (∼ χ → ⨆ (∼ Φ))› modus-ponens by auto
  qed
  then show ?case
    using biconditional-transitivity-rule list-curry-uncurry by blast
qed
```

5.9.5 Monotony of Conjunction and Disjunction

```
lemma (in classical-logic) conjunction-monotonic-identity:
  ⊢ (φ → ψ) → (φ ⊓ χ) → (ψ ⊓ χ)
  unfolding conjunction-def
  using modus-ponens
        flip-hypothetical-syllogism
  by blast

lemma (in classical-logic) conjunction-monotonic:
  assumes ⊢ φ → ψ
  shows ⊢ (φ ⊓ χ) → (ψ ⊓ χ)
  using assms
        modus-ponens
        conjunction-monotonic-identity
```

by *blast*

lemma (**in** *classical-logic*) *disjunction-monotonic-identity*:
$\vdash (\varphi \to \psi) \to (\varphi \sqcup \chi) \to (\psi \sqcup \chi)$
unfolding *disjunction-def*
using *modus-ponens*
flip-hypothetical-syllogism
by *blast*

lemma (**in** *classical-logic*) *disjunction-monotonic*:
assumes $\vdash \varphi \to \psi$
shows $\vdash (\varphi \sqcup \chi) \to (\psi \sqcup \chi)$
using *assms*
modus-ponens
disjunction-monotonic-identity
by *blast*

5.9.6 Distribution Identities

lemma (**in** *classical-logic*) *conjunction-distribution*:
$\vdash ((\psi \sqcup \chi) \sqcap \varphi) \leftrightarrow ((\psi \sqcap \varphi) \sqcup (\chi \sqcap \varphi))$
proof −
have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} (((\langle\psi\rangle \sqcup \langle\chi\rangle) \sqcap \langle\varphi\rangle) \leftrightarrow ((\langle\psi\rangle \sqcap \langle\varphi\rangle) \sqcup (\langle\chi\rangle \sqcap \langle\varphi\rangle)))$
by *auto*
hence $\vdash (\!|\ ((\langle\psi\rangle \sqcup \langle\chi\rangle) \sqcap \langle\varphi\rangle) \leftrightarrow ((\langle\psi\rangle \sqcap \langle\varphi\rangle) \sqcup (\langle\chi\rangle \sqcap \langle\varphi\rangle))\ |\!)$
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *subtraction-distribution*:
$\vdash ((\psi \sqcup \chi) \setminus \varphi) \leftrightarrow ((\psi \setminus \varphi) \sqcup (\chi \setminus \varphi))$
by (*simp add*: *conjunction-distribution subtraction-def*)

lemma (**in** *classical-logic*) *conjunction-arbitrary-distribution*:
$\vdash (\bigsqcup\ \Psi \sqcap \varphi) \leftrightarrow \bigsqcup\ [\psi \sqcap \varphi.\ \psi \leftarrow \Psi]$
proof (*induct* Ψ)
case *Nil*
then show *?case*
by (*simp add*: *ex-falso-quodlibet*
biconditional-def
conjunction-left-elimination)
next
case (*Cons* ψ Ψ)
have $\vdash (\bigsqcup\ (\psi \# \Psi) \sqcap \varphi) \leftrightarrow ((\psi \sqcap \varphi) \sqcup ((\bigsqcup\ \Psi) \sqcap \varphi))$
using *conjunction-distribution* **by** *auto*
moreover
from *Cons* **have**
$\vdash ((\psi \sqcap \varphi) \sqcup ((\bigsqcup\ \Psi) \sqcap \varphi)) \leftrightarrow ((\psi \sqcap \varphi) \sqcup (\bigsqcup\ [\psi \sqcap \varphi.\ \psi \leftarrow \Psi]))$
unfolding *disjunction-def biconditional-def*

by (*simp*, *meson modus-ponens hypothetical-syllogism*)
ultimately show *?case*
by (*simp*, *metis biconditional-transitivity-rule*)
qed

lemma (**in** *classical-logic*) *subtraction-arbitrary-distribution*:
$\vdash (\bigsqcup\ \Psi \setminus \varphi) \leftrightarrow \bigsqcup\ [\psi \setminus \varphi.\ \psi \leftarrow \Psi]$
by (*simp add*: *conjunction-arbitrary-distribution subtraction-def*)

lemma (**in** *classical-logic*) *disjunction-distribution*:
$\vdash (\varphi \sqcup (\psi \sqcap \chi)) \leftrightarrow ((\varphi \sqcup \psi) \sqcap (\varphi \sqcup \chi))$
proof −
have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} (\langle\varphi\rangle \sqcup (\langle\psi\rangle \sqcap \langle\chi\rangle)) \leftrightarrow ((\langle\varphi\rangle \sqcup \langle\psi\rangle) \sqcap (\langle\varphi\rangle \sqcup \langle\chi\rangle))$
by *auto*
hence $\vdash (\!|\ (\langle\varphi\rangle \sqcup (\langle\psi\rangle \sqcap \langle\chi\rangle)) \leftrightarrow ((\langle\varphi\rangle \sqcup \langle\psi\rangle) \sqcap (\langle\varphi\rangle \sqcup \langle\chi\rangle))\ |\!)$
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *implication-distribution*:
$\vdash (\varphi \rightarrow (\psi \sqcap \chi)) \leftrightarrow ((\varphi \rightarrow \psi) \sqcap (\varphi \rightarrow \chi))$
proof −
have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} (\langle\varphi\rangle \rightarrow (\langle\psi\rangle \sqcap \langle\chi\rangle)) \leftrightarrow ((\langle\varphi\rangle \rightarrow \langle\psi\rangle) \sqcap (\langle\varphi\rangle \rightarrow \langle\chi\rangle))$
by *auto*
hence $\vdash (\!|\ (\langle\varphi\rangle \rightarrow (\langle\psi\rangle \sqcap \langle\chi\rangle)) \leftrightarrow ((\langle\varphi\rangle \rightarrow \langle\psi\rangle) \sqcap (\langle\varphi\rangle \rightarrow \langle\chi\rangle))\ |\!)$
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *list-implication-distribution*:
$\vdash (\Phi :\rightarrow (\psi \sqcap \chi)) \leftrightarrow ((\Phi :\rightarrow \psi) \sqcap (\Phi :\rightarrow \chi))$
proof (*induct* Φ)
case *Nil*
then show *?case*
by (*simp add*: *biconditional-reflection*)
next
case (*Cons* φ Φ)
hence $\vdash (\varphi \# \Phi) :\rightarrow (\psi \sqcap \chi) \leftrightarrow (\varphi \rightarrow (\Phi :\rightarrow \psi \sqcap \Phi :\rightarrow \chi))$
by (*metis*
modus-ponens
biconditional-def
hypothetical-syllogism
list-implication.simps(*2*)
weak-conjunction-deduction-equivalence)
moreover
have $\vdash (\varphi \rightarrow (\Phi :\rightarrow \psi \sqcap \Phi :\rightarrow \chi)) \leftrightarrow (((\varphi \# \Phi) :\rightarrow \psi) \sqcap ((\varphi \# \Phi) :\rightarrow \chi))$
using *implication-distribution* **by** *auto*
ultimately show *?case*
by (*simp*, *metis biconditional-transitivity-rule*)

```
qed

lemma (in classical-logic) biconditional-conjunction-weaken:
  ⊢ (α ↔ β) → ((γ ⊓ α) ↔ (γ ⊓ β))
proof –
  have ∀ 𝔐. 𝔐 ⊨prop (⟨α⟩ ↔ ⟨β⟩) → ((⟨γ⟩ ⊓ ⟨α⟩) ↔ (⟨γ⟩ ⊓ ⟨β⟩))
      by auto
  hence ⊢ ⦇ (⟨α⟩ ↔ ⟨β⟩) → ((⟨γ⟩ ⊓ ⟨α⟩) ↔ (⟨γ⟩ ⊓ ⟨β⟩)) ⦈
    using propositional-semantics by blast
  thus ?thesis by simp
qed

lemma (in classical-logic) biconditional-conjunction-weaken-rule:
  ⊢ (α ↔ β) ⟹ ⊢ (γ ⊓ α) ↔ (γ ⊓ β)
  using modus-ponens biconditional-conjunction-weaken by blast

lemma (in classical-logic) disjunction-arbitrary-distribution:
  ⊢ (φ ⊔ ⨅ Ψ) ↔ ⨅ [φ ⊔ ψ. ψ ← Ψ]
proof (induct Ψ)
  case Nil
  then show ?case
    unfolding disjunction-def biconditional-def
    using axiom-k modus-ponens verum-tautology
    by (simp, blast)
next
  case (Cons ψ Ψ)
  have ⊢ (φ ⊔ ⨅ (ψ # Ψ)) ↔ ((φ ⊔ ψ) ⊓ (φ ⊔ ⨅ Ψ))
    by (simp add: disjunction-distribution)
  moreover
  from biconditional-conjunction-weaken-rule
       Cons
  have ⊢ ((φ ⊔ ψ) ⊓ φ ⊔ ⨅ Ψ) ↔ ⨅ (map (λ χ . φ ⊔ χ) (ψ # Ψ))
    by simp
  ultimately show ?case
    by (metis biconditional-transitivity-rule)
qed

lemma (in classical-logic) list-implication-arbitrary-distribution:
  ⊢ (Φ :→ ⨅ Ψ) ↔ ⨅ [Φ :→ ψ. ψ ← Ψ]
proof (induct Ψ)
  case Nil
  then show ?case
    by (simp add: biconditional-def,
        meson
          axiom-k
          modus-ponens
          list-implication-axiom-k
          verum-tautology)
next
```

case (*Cons* ψ Ψ)
have $\vdash \Phi :\rightarrow \bigsqcap (\psi \# \Psi) \leftrightarrow (\Phi :\rightarrow \psi \sqcap \Phi :\rightarrow \bigsqcap \Psi)$
using *list-implication-distribution*
by *fastforce*
moreover
from *biconditional-conjunction-weaken-rule*
Cons
have $\vdash (\Phi :\rightarrow \psi \sqcap \Phi :\rightarrow \bigsqcap \Psi) \leftrightarrow \bigsqcap [\Phi :\rightarrow \psi.\ \psi \leftarrow (\psi \# \Psi)]$
by *simp*
ultimately show *?case*
by (*metis biconditional-transitivity-rule*)
qed

lemma (**in** *classical-logic*) *implication-arbitrary-distribution*:
$\vdash (\varphi \rightarrow \bigsqcap \Psi) \leftrightarrow \bigsqcap [\varphi \rightarrow \psi.\ \psi \leftarrow \Psi]$
using *list-implication-arbitrary-distribution* [**where** *?*Φ=[φ]]
by *simp*

5.9.7 Negation

lemma (**in** *classical-logic*) *double-negation-biconditional*:
$\vdash \sim (\sim \varphi) \leftrightarrow \varphi$
unfolding *biconditional-def negation-def*
by (*simp add*: *double-negation double-negation-converse*)

lemma (**in** *classical-logic*) *double-negation-elimination* [*simp*]:
$\Gamma \Vdash \sim (\sim \varphi) = \Gamma \Vdash \varphi$
using
set-deduction-weaken
biconditional-weaken
double-negation-biconditional
by *metis*

lemma (**in** *classical-logic*) *alt-double-negation-elimination* [*simp*]:
$\Gamma \Vdash (\varphi \rightarrow \bot) \rightarrow \bot \equiv \Gamma \Vdash \varphi$
using *double-negation-elimination*
unfolding *negation-def*
by *auto*

lemma (**in** *classical-logic*) *base-double-negation-elimination* [*simp*]:
$\vdash \sim (\sim \varphi) = \vdash \varphi$
by (*metis double-negation-elimination set-deduction-base-theory*)

lemma (**in** *classical-logic*) *alt-base-double-negation-elimination* [*simp*]:
$\vdash (\varphi \rightarrow \bot) \rightarrow \bot \equiv \vdash \varphi$
using *base-double-negation-elimination*
unfolding *negation-def*
by *auto*

5.9.8 Mutual Exclusion Identities

lemma (**in** *classical-logic*) *exclusion-contrapositive-equivalence*:
  $\vdash (\varphi \to \gamma) \leftrightarrow \sim (\varphi \sqcap \sim \gamma)$
proof −
  have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} (\langle\varphi\rangle \to \langle\gamma\rangle) \leftrightarrow \sim (\langle\varphi\rangle \sqcap \sim \langle\gamma\rangle)$
    by *auto*
  hence $\vdash (\!| (\langle\varphi\rangle \to \langle\gamma\rangle) \leftrightarrow \sim (\langle\varphi\rangle \sqcap \sim \langle\gamma\rangle) |\!)$
    using *propositional-semantics* **by** *blast*
  thus *?thesis* **by** *simp*
qed

lemma (**in** *classical-logic*) *disjuction-exclusion-equivalence*:
  $\Gamma \Vdash \sim (\psi \sqcap \bigsqcup\ \Phi) \equiv \forall\ \varphi \in set\ \Phi.\ \Gamma \Vdash \sim (\psi \sqcap \varphi)$
proof (*induct* Φ)
  case *Nil*
  then show *?case*
    by (*simp add*:
        conjunction-right-elimination
        negation-def
        set-deduction-weaken)
next
  case (*Cons* φ Φ)
  have $\vdash \sim (\psi \sqcap \bigsqcup\ (\varphi \# \Phi)) \leftrightarrow \sim (\psi \sqcap (\varphi \sqcup \bigsqcup\ \Phi))$
    by (*simp add*: *biconditional-reflection*)
  moreover have $\vdash \sim (\psi \sqcap (\varphi \sqcup \bigsqcup\ \Phi)) \leftrightarrow (\sim (\psi \sqcap \varphi) \sqcap \sim (\psi \sqcap \bigsqcup\ \Phi))$
  proof −
    have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} \sim (\langle\psi\rangle \sqcap (\langle\varphi\rangle \sqcup \langle\bigsqcup\ \Phi\rangle))$
                $\leftrightarrow (\sim (\langle\psi\rangle \sqcap \langle\varphi\rangle) \sqcap \sim (\langle\psi\rangle \sqcap \langle\bigsqcup\ \Phi\rangle))$
      by *auto*
    hence $\vdash (\!| \sim (\langle\psi\rangle \sqcap (\langle\varphi\rangle \sqcup \langle\bigsqcup\ \Phi\rangle))$
            $\leftrightarrow (\sim (\langle\psi\rangle \sqcap \langle\varphi\rangle) \sqcap \sim (\langle\psi\rangle \sqcap \langle\bigsqcup\ \Phi\rangle)) |\!)$
      using *propositional-semantics* **by** *blast*
    thus *?thesis* **by** *simp*
  qed
  ultimately
  have $\vdash \sim (\psi \sqcap \bigsqcup\ (\varphi \# \Phi)) \leftrightarrow (\sim (\psi \sqcap \varphi) \sqcap \sim (\psi \sqcap \bigsqcup\ \Phi))$
    by *simp*
  hence $\Gamma \Vdash \sim (\psi \sqcap \bigsqcup\ (\varphi \# \Phi)) = (\Gamma \Vdash \sim (\psi \sqcap \varphi)$
          $\wedge (\forall \varphi{\in}set\ \Phi.\ \Gamma \Vdash \sim (\psi \sqcap \varphi)))$
    using *set-deduction-weaken* [**where** Γ=Γ]
        conjunction-set-deduction-equivalence [**where** Γ=Γ]
        Cons.hyps
        biconditional-def
        set-deduction-modus-ponens
    by *metis*
  thus $\Gamma \Vdash \sim (\psi \sqcap \bigsqcup\ (\varphi \# \Phi)) = (\forall \varphi{\in}set\ (\varphi \# \Phi).\ \Gamma \Vdash \sim (\psi \sqcap \varphi))$
    by *simp*
qed

```
lemma (in classical-logic) exclusive-elimination1:
  assumes Γ ⊩ ∐ Φ
  shows ∀ φ ∈ set Φ. ∀ ψ ∈ set Φ. (φ ≠ ψ) ⟶ Γ ⊩ ∼ (φ ⊓ ψ)
  using assms
proof (induct Φ)
  case Nil
  thus ?case by auto
next
  case (Cons χ Φ)
  assume Γ ⊩ ∐ (χ # Φ)
  hence Γ ⊩ ∐ Φ by simp
  hence ∀φ∈set Φ. ∀ψ∈set Φ. φ ≠ ψ ⟶ Γ ⊩ ∼ (φ ⊓ ψ)
    using Cons.hyps by blast
  moreover have Γ ⊩ ∼ (χ ⊓ ⨆ Φ)
    using ‹Γ ⊩ ∐ (χ # Φ)› conjunction-set-deduction-equivalence by auto
  hence ∀ φ ∈ set Φ. Γ ⊩ ∼ (χ ⊓ φ)
    using disjuction-exclusion-equivalence by auto
  moreover {
    fix φ
    have ⊢ ∼ (χ ⊓ φ) → ∼ (φ ⊓ χ)
      unfolding negation-def
                conjunction-def
      using modus-ponens flip-hypothetical-syllogism flip-implication by blast
  }
  with ‹∀ φ ∈ set Φ. Γ ⊩ ∼ (χ ⊓ φ)› have ∀ φ ∈ set Φ. Γ ⊩ ∼ (φ ⊓ χ)
    using set-deduction-weaken [where Γ=Γ]
          set-deduction-modus-ponens [where Γ=Γ]
    by blast
  ultimately
  show ∀φ ∈ set (χ # Φ). ∀ψ ∈ set (χ # Φ). φ ≠ ψ ⟶ Γ ⊩ ∼ (φ ⊓ ψ)
    by simp
qed

lemma (in classical-logic) exclusive-elimination2:
  assumes Γ ⊩ ∐ Φ
  shows ∀ φ ∈ duplicates Φ. Γ ⊩ ∼ φ
  using assms
proof (induct Φ)
  case Nil
  then show ?case by simp
next
  case (Cons φ Φ)
  assume Γ ⊩ ∐ (φ # Φ)
  hence Γ ⊩ ∐ Φ by simp
  hence ∀φ∈duplicates Φ. Γ ⊩ ∼ φ using Cons.hyps by auto
  show ?case
  proof cases
    assume φ ∈ set Φ
    moreover {
```

fix $\varphi\ \psi\ \chi$
have $\vdash \sim (\varphi \sqcap (\psi \sqcup \chi)) \leftrightarrow (\sim (\varphi \sqcap \psi) \sqcap \sim (\varphi \sqcap \chi))$
proof −
have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} \sim (\langle\varphi\rangle \sqcap (\langle\psi\rangle \sqcup \langle\chi\rangle))$
$\leftrightarrow (\sim (\langle\varphi\rangle \sqcap \langle\psi\rangle) \sqcap \sim (\langle\varphi\rangle \sqcap \langle\chi\rangle))$
by *auto*
hence $\vdash (\!| \sim (\langle\varphi\rangle \sqcap (\langle\psi\rangle \sqcup \langle\chi\rangle)) \leftrightarrow (\sim (\langle\varphi\rangle \sqcap \langle\psi\rangle) \sqcap \sim (\langle\varphi\rangle \sqcap \langle\chi\rangle)) |\!)$
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
hence $\Gamma \Vdash \sim (\varphi \sqcap (\psi \sqcup \chi)) \equiv \Gamma \Vdash \sim (\varphi \sqcap \psi) \sqcap \sim (\varphi \sqcap \chi)$
using *set-deduction-weaken*
biconditional-weaken **by** *presburger*
}
moreover
have $\vdash \sim (\varphi \sqcap \varphi) \leftrightarrow \sim \varphi$
proof −
have $\forall\ \mathfrak{M}.\ \mathfrak{M} \models_{prop} \sim (\langle\varphi\rangle \sqcap \langle\varphi\rangle) \leftrightarrow \sim \langle\varphi\rangle$
by *auto*
hence $\vdash (\!| \sim (\langle\varphi\rangle \sqcap \langle\varphi\rangle) \leftrightarrow \sim \langle\varphi\rangle |\!)$
using *propositional-semantics* **by** *blast*
thus *?thesis* **by** *simp*
qed
hence $\Gamma \Vdash \sim (\varphi \sqcap \varphi) \equiv \Gamma \Vdash \sim \varphi$
using *set-deduction-weaken*
biconditional-weaken **by** *presburger*
moreover have $\Gamma \Vdash \sim (\varphi \sqcap \bigsqcup\ \Phi)$ **using** ‹$\Gamma \Vdash \coprod\ (\varphi \# \Phi)$› **by** *simp*
ultimately have $\Gamma \Vdash \sim \varphi$ **by** (*induct* Φ, *simp*, *simp*, *blast*)
thus *?thesis* **using** ‹$\varphi \in$ *set* Φ› ‹$\forall \varphi \in$ *duplicates* $\Phi.\ \Gamma \Vdash \sim \varphi$› **by** *simp*
next
assume $\varphi \notin$ *set* Φ
hence *duplicates* $(\varphi \# \Phi)$ = *duplicates* Φ **by** *simp*
then show *?thesis* **using** ‹$\forall \varphi \in$ *duplicates* $\Phi.\ \Gamma \Vdash \sim \varphi$›
by *auto*
qed
qed

lemma (**in** *classical-logic*) *exclusive-equivalence*:
$\Gamma \Vdash \coprod\ \Phi =$
$((\forall \varphi \in$ *duplicates* $\Phi.\ \Gamma \Vdash \sim \varphi) \wedge$
$(\forall\ \varphi \in$ *set* $\Phi.\ \forall\ \psi \in$ *set* $\Phi.\ (\varphi \neq \psi) \longrightarrow \Gamma \Vdash \sim (\varphi \sqcap \psi)))$
proof −
{
assume $\forall \varphi \in$ *duplicates* $\Phi.\ \Gamma \Vdash \sim \varphi$
$\forall\ \varphi \in$ *set* $\Phi.\ \forall\ \psi \in$ *set* $\Phi.\ (\varphi \neq \psi) \longrightarrow \Gamma \Vdash \sim (\varphi \sqcap \psi)$
hence $\Gamma \Vdash \coprod\ \Phi$
proof (*induct* Φ)
case *Nil*
then show *?case*

```
    by (simp add: set-deduction-weaken)
next
  case (Cons φ Φ)
  assume A: ∀φ∈duplicates (φ # Φ). Γ ⊩ ∼ φ
     and B: ∀χ∈set (φ # Φ). ∀ψ∈set (φ # Φ). χ ≠ ψ ⟶ Γ ⊩ ∼ (χ ⊓ ψ)
  hence C: Γ ⊩ ∐ Φ using Cons.hyps by simp
  then show ?case
  proof cases
    assume φ ∈ duplicates (φ # Φ)
    moreover from this have Γ ⊩ ∼ φ using A by auto
    moreover have duplicates Φ ⊆ set Φ by (induct Φ, simp, auto)
    ultimately have φ ∈ set Φ by (metis duplicates.simps(2) subsetCE)
    hence ⊢ ∼ φ ↔ ∼(φ ⊓ ⨆ Φ)
    proof (induct Φ)
      case Nil
      then show ?case by simp
    next
      case (Cons ψ Φ)
      assume φ ∈ set (ψ # Φ)
      then show ⊢ ∼ φ ↔ ∼ (φ ⊓ ⨆ (ψ # Φ))
      proof −
        {
          assume φ = ψ
          hence ?thesis
          proof −
            have ∀ 𝔐. 𝔐 ⊨prop ∼ ⟨φ⟩ ↔ ∼ (⟨φ⟩ ⊓ (⟨ψ⟩ ⊔ ⟨⨆ Φ⟩))
              using ‹φ = ψ› by auto
            hence ⊢ ⦇ ∼ ⟨φ⟩ ↔ ∼ (⟨φ⟩ ⊓ (⟨ψ⟩ ⊔ ⟨⨆ Φ⟩)) ⦈
              using propositional-semantics by blast
            thus ?thesis by simp
          qed
        }
        moreover
        {
          assume φ ≠ ψ
          hence φ ∈ set Φ using ‹φ ∈ set (ψ # Φ)› by auto
          hence ⊢ ∼ φ ↔ ∼ (φ ⊓ ⨆ Φ) using Cons.hyps by auto
          moreover have ⊢ (∼ φ ↔ ∼ (φ ⊓ ⨆ Φ))
                      → (∼ φ ↔ ∼ (φ ⊓ (ψ ⊔ ⨆ Φ)))
          proof −
            have ∀ 𝔐. 𝔐 ⊨prop (∼ ⟨φ⟩ ↔ ∼ (⟨φ⟩ ⊓ ⟨⨆ Φ⟩)) →
                        (∼ ⟨φ⟩ ↔ ∼ (⟨φ⟩ ⊓ (⟨ψ⟩ ⊔ ⟨⨆ Φ⟩)))
              by auto
            hence ⊢ ⦇ (∼ ⟨φ⟩ ↔ ∼ (⟨φ⟩ ⊓ ⟨⨆ Φ⟩))
                    → (∼ ⟨φ⟩ ↔ ∼ (⟨φ⟩ ⊓ (⟨ψ⟩ ⊔ ⟨⨆ Φ⟩))) ⦈
              using propositional-semantics by blast
            thus ?thesis by simp
          qed
          ultimately have ?thesis using modus-ponens by simp
```

          }
          ultimately show *?thesis* **by** *auto*
        qed
      qed
      with ‹Γ ⊩ ∼ φ› **have** Γ ⊩ ∼(φ ⊓ ⨆ Φ)
        using *biconditional-weaken set-deduction-weaken* **by** *blast*
      with ‹Γ ⊩ ∐ Φ› **show** *?thesis* **by** *simp*
    next
      assume φ ∉ *duplicates* (φ # Φ)
      hence φ ∉ *set* Φ **by** *auto*
      with *B* **have** ∀ψ∈*set* Φ. Γ ⊩ ∼ (φ ⊓ ψ) **by** (*simp*, *metis*)
      hence Γ ⊩ ∼ (φ ⊓ ⨆ Φ)
        by (*simp add*: *disjuction-exclusion-equivalence*)
      with ‹Γ ⊩ ∐ Φ› **show** *?thesis* **by** *simp*
    qed
  qed
}
thus *?thesis*
  by (*metis exclusive-elimination1 exclusive-elimination2*)
qed

5.9.9 Miscellaneous Disjunctive Normal Form Identities

lemma (**in** *classical-logic*) *map-negation-list-implication*:
⊢ ((**∼** Φ) :→ (∼ φ)) ↔ (φ → ⨆ Φ)
proof (*induct* Φ)
  case *Nil*
  then show *?case*
    unfolding
      biconditional-def
      map-negation-def
      negation-def
    using
      conjunction-introduction
      modus-ponens
      trivial-implication
    by *simp*
next
  case (*Cons* ψ Φ)
  have ⊢ (**∼** Φ :→ ∼ φ ↔ (φ → ⨆ Φ))
        → (∼ ψ → **∼** Φ :→ ∼ φ) ↔ (φ → (ψ ⊔ ⨆ Φ))
  proof −
    have ∀𝔐. 𝔐 $\models_{prop}$ (⟨**∼** Φ :→ ∼ φ⟩ ↔ (⟨φ⟩ → ⟨⨆ Φ⟩)) →
                (∼ ⟨ψ⟩ → ⟨**∼** Φ :→ ∼ φ⟩) ↔ (⟨φ⟩ → (⟨ψ⟩ ⊔ ⟨⨆ Φ⟩))
      by *fastforce*
    hence ⊢ ⦇ (⟨**∼** Φ :→ ∼ φ⟩ ↔ (⟨φ⟩ → ⟨⨆ Φ⟩)) →
              (∼ ⟨ψ⟩ → ⟨**∼** Φ :→ ∼ φ⟩) ↔ (⟨φ⟩ → (⟨ψ⟩ ⊔ ⟨⨆ Φ⟩)) ⦈
      using *propositional-semantics* **by** *blast*
    thus *?thesis*

by *simp*
qed
with *Cons* **show** *?case*
by (*metis*
map-negation-def
list.simps(*9*)
arbitrary-disjunction.simps(*2*)
modus-ponens
list-implication.simps(*2*))
qed

lemma (**in** *classical-logic*) *conj-dnf-distribute*:
$\vdash \bigsqcup\ (map\ (\bigsqcap \circ (\lambda\ \varphi s.\ \varphi \# \varphi s))\ \Lambda) \leftrightarrow (\varphi \sqcap \bigsqcup\ (map \bigsqcap\ \Lambda))$
proof(*induct* Λ)
case *Nil*
have $\vdash \bot \leftrightarrow (\varphi \sqcap \bot)$
proof −
let $?\varphi = \bot \leftrightarrow (\langle\varphi\rangle \sqcap \bot)$
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* Ψ Λ)
assume $\vdash \bigsqcup\ (map\ (\bigsqcap \circ (\lambda\ \varphi s.\ \varphi \# \varphi s))\ \Lambda) \leftrightarrow (\varphi \sqcap \bigsqcup\ (map \bigsqcap\ \Lambda))$
(**is** $\vdash\ ?A \leftrightarrow (\varphi \sqcap\ ?B)$)
moreover
have $\vdash (?A \leftrightarrow (\varphi \sqcap\ ?B)) \to (((\varphi \sqcap \bigsqcap\ \Psi) \sqcup\ ?A) \leftrightarrow (\varphi \sqcap \bigsqcap\ \Psi \sqcup\ ?B))$
proof −
let $?\varphi = (\langle ?A\rangle \leftrightarrow (\langle\varphi\rangle \sqcap \langle ?B\rangle)) \to$
$(((\langle\varphi\rangle \sqcap \langle\bigsqcap\ \Psi\rangle) \sqcup \langle ?A\rangle) \leftrightarrow (\langle\varphi\rangle \sqcap \langle\bigsqcap\ \Psi\rangle \sqcup \langle ?B\rangle))$
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 $\vdash ((\varphi \sqcap \bigsqcap\ \Psi) \sqcup\ ?A) \leftrightarrow (\varphi \sqcap \bigsqcap\ \Psi \sqcup\ ?B)$
using *modus-ponens*
by *blast*
moreover
have $map\ (\bigsqcap \circ (\lambda\ \varphi s.\ \varphi \# \varphi s))\ \Lambda = map\ (\lambda\Psi.\ \varphi \sqcap \bigsqcap\ \Psi)\ \Lambda$ **by** *simp*
ultimately show *?case* **by** *simp*
qed

lemma (**in** *classical-logic*) *append-dnf-distribute*:
$\vdash \bigsqcup\ (map\ (\bigsqcap \circ (\lambda\ \Psi.\ \Phi\ @\ \Psi))\ \Lambda) \leftrightarrow (\bigsqcap\ \Phi \sqcap \bigsqcup\ (map \bigsqcap\ \Lambda))$
proof(*induct* Φ)

case *Nil*
have $\vdash \bigsqcup$ (*map* $\bigsqcap$ Λ) $\leftrightarrow$ ($\top \sqcap \bigsqcup$ (*map* $\bigsqcap$ Λ))
(**is** $\vdash$ *?A* $\leftrightarrow$ ($\top \sqcap$ *?A*))
proof −
let $?\varphi = \langle ?A \rangle \leftrightarrow ((\bot \to \bot) \sqcap \langle ?A \rangle)$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *simp*
hence $\vdash (\!|\ ?\varphi\ |\!)$ **using** *propositional-semantics* **by** *blast*
thus *?thesis*
unfolding *verum-def*
by *simp*
qed
then show *?case* **by** *simp*
next
case (*Cons* φ Φ)
have $\vdash \bigsqcup$ (*map* ($\bigsqcap \circ$ (@) Φ) Λ) $\leftrightarrow$ ($\bigsqcap$ $\Phi \sqcap \bigsqcup$ (*map* $\bigsqcap$ Λ))
= $\vdash \bigsqcup$ (*map* $\bigsqcap$ (*map* ((@) Φ) Λ)) $\leftrightarrow$ ($\bigsqcap$ $\Phi \sqcap \bigsqcup$ (*map* $\bigsqcap$ Λ))
by *simp*
with *Cons* **have**
$\vdash \bigsqcup$ (*map* $\bigsqcap$ (*map* (λ Ψ. Φ @ Ψ) Λ)) $\leftrightarrow$ ($\bigsqcap$ $\Phi \sqcap \bigsqcup$ (*map* $\bigsqcap$ Λ))
(**is** $\vdash \bigsqcup$ (*map* $\bigsqcap$ *?A*) $\leftrightarrow$ (*?B* $\sqcap$ *?C*))
by *meson*
moreover have $\vdash \bigsqcup$ (*map* $\bigsqcap$ *?A*) $\leftrightarrow$ (*?B* $\sqcap$ *?C*)
$\to$ ($\bigsqcup$ (*map* ($\bigsqcap \circ$ (λ φs. φ # φs)) *?A*) $\leftrightarrow$ ($\varphi \sqcap \bigsqcup$ (*map* $\bigsqcap$ *?A*)))
$\to \bigsqcup$ (*map* ($\bigsqcap \circ$ (λ φs. φ # φs)) *?A*) $\leftrightarrow$ (($\varphi \sqcap$ *?B*) $\sqcap$ *?C*)
proof −
let $?\varphi = \langle \bigsqcup$ (*map* $\bigsqcap$ *?A*)$\rangle \leftrightarrow (\langle ?B \rangle \sqcap \langle ?C \rangle)$
$\to$ ($\langle \bigsqcup$ (*map* ($\bigsqcap \circ$ (λ φs. φ # φs)) *?A*)$\rangle \leftrightarrow (\langle \varphi \rangle \sqcap \langle \bigsqcup$ (*map* $\bigsqcap$ *?A*)$\rangle$))
$\to \langle \bigsqcup$ (*map* ($\bigsqcap \circ$ (λ φs. φ # φs)) *?A*)$\rangle \leftrightarrow ((\langle \varphi \rangle \sqcap \langle ?B \rangle) \sqcap \langle ?C \rangle)$
have $\forall \mathfrak{M}.\ \mathfrak{M} \models_{prop} ?\varphi$ **by** *simp*
hence $\vdash (\!|\ ?\varphi\ |\!)$ **using** *propositional-semantics* **by** *blast*
thus *?thesis*
by *simp*
qed
ultimately have $\vdash \bigsqcup$ (*map* ($\bigsqcap \circ$ (λ φs. φ # φs)) *?A*) $\leftrightarrow$ (($\varphi \sqcap$ *?B*) $\sqcap$ *?C*)
using *modus-ponens conj-dnf-distribute*
by *blast*
moreover
have $\bigsqcap \circ$ (@) (φ # Φ) = $\bigsqcap \circ$ (#) $\varphi \circ$ (@) Φ **by** *auto*
hence
$\vdash \bigsqcup$ (*map* ($\bigsqcap \circ$ (@) (φ # Φ)) Λ) $\leftrightarrow$ ($\bigsqcap$ (φ # Φ) $\sqcap$ *?C*)
= $\vdash \bigsqcup$ (*map* ($\bigsqcap \circ$ (#) φ) *?A*) $\leftrightarrow$ (($\varphi \sqcap$ *?B*) $\sqcap$ *?C*)
by *simp*
ultimately show *?case* **by** *meson*
qed

unbundle *funcset-syntax*

end

Bibliography

[1] P. Blackburn, M. de Rijke, and Y. Venema. Section 4.2 Canonical Models. In *Modal Logic*, pages 196–201.

[2] A. Bobenrieth. The Origins of the Use of the Argument of Trivialization in the Twentieth Century. 31(2):111–121.

[3] G. Boolos. Don't Eliminate Cut. 13(4):373–378.

[4] G. Gentzen. Untersuchungen über das logische schlieSSen. i. 39(1):176–210.

[5] C. S. Peirce. On the Algebra of Logic: A Contribution to the Philosophy of Notation. 7(2):180–196.

[6] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Number 43 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2nd ed edition.

[7] A. Urquhart. Implicational Formulas in Intuitionistic Logic. 39(4):661–664.

[8] D. van Dalen. *Logic and Structure*. Universitext. Springer-Verlag, 5 edition.