Epistemic Logic

Asta Halkjær From

February 23, 2021

Abstract

This work is a formalization of epistemic logic with countably many agents. It includes proofs of soundness and completeness for the axiom system K. The completeness proof is based on the textbook "Reasoning About Knowledge" by Fagin, Halpern, Moses and Vardi (MIT Press 1995) [1].

Contents

theory *Epistemic-Logic* **imports** *HOL−Library.Countable* **begin**

1 Syntax

type-synonym *id* = *string*

datatype *'i fm*
= *FF* (⊥)
| *Pro id*
| *Dis* ‹*'i fm*› ‹*'i fm*› (**infixr** ∨ *30*)
| *Con* ‹*'i fm*› ‹*'i fm*› (**infixr** ∧ *35*)
| *Imp* ‹*'i fm*› ‹*'i fm*› (**infixr** ⟶ *25*)
| *K 'i* ‹*'i fm*›

abbreviation *TT* (⊤) **where**
‹*TT* ≡ ⊥ ⟶ ⊥›

abbreviation *Neg* (¬ - [*40*] *40*) **where**
‹*Neg p* ≡ *p* ⟶ ⊥›

2 Semantics

datatype (*'i*, *'s*) *kripke* = *Kripke* (*π*: ‹*'s* ⇒ *id* ⇒ *bool*›) (*𝒦*: ‹*'i* ⇒ *'s* ⇒ *'s set*›)

primrec *semantics* :: ‹(*'i*, *'s*) *kripke* ⇒ *'s* ⇒ *'i fm* ⇒ *bool*›
(-, - ⊨ - [*50*,*50*] *50*) **where**
‹(-, - ⊨ ⊥) = *False*›
| ‹(*M*, *s* ⊨ *Pro i*) = *π M s i*›
| ‹(*M*, *s* ⊨ (*p* ∨ *q*)) = ((*M*, *s* ⊨ *p*) ∨ (*M*, *s* ⊨ *q*))›
| ‹(*M*, *s* ⊨ (*p* ∧ *q*)) = ((*M*, *s* ⊨ *p*) ∧ (*M*, *s* ⊨ *q*))›
| ‹(*M*, *s* ⊨ (*p* ⟶ *q*)) = ((*M*, *s* ⊨ *p*) ⟶ (*M*, *s* ⊨ *q*))›
| ‹(*M*, *s* ⊨ *K i p*) = (∀ *t* ∈ *𝒦 M i s*. *M*, *t* ⊨ *p*)›

3 Utility

abbreviation *reflexive* :: ‹(*'i*, *'s*) *kripke* ⇒ *bool*› **where**
‹*reflexive M* ≡ ∀ *i s*. *s* ∈ *𝒦 M i s*›

abbreviation *symmetric* :: ‹(*'i*, *'s*) *kripke* ⇒ *bool*› **where**
‹*symmetric M* ≡ ∀ *i s t*. *t* ∈ *𝒦 M i s* ⟷ *s* ∈ *𝒦 M i t*›

abbreviation *transitive* :: ‹(*'i*, *'s*) *kripke* ⇒ *bool*› **where**
‹*transitive M* ≡ ∀ *i s t u*. *t* ∈ *𝒦 M i s* ∧ *u* ∈ *𝒦 M i t* ⟶ *u* ∈ *𝒦 M i s*›

lemma *Imp-intro*: ‹(*M*, *s* ⊨ *p* ⟹ *M*, *s* ⊨ *q*) ⟹ *M*, *s* ⊨ *Imp p q*›
by *simp*

4 S5 Axioms

theorem *distribution*: ‹M, $s \models (K\ i\ p \wedge K\ i\ (p \longrightarrow q) \longrightarrow K\ i\ q)$›
proof (*rule Imp-intro*)
  assume ‹M, $s \models (K\ i\ p \wedge K\ i\ (p \longrightarrow q))$›
  then have ‹M, $s \models K\ i\ p$› ‹M, $s \models K\ i\ (p \longrightarrow q)$›
    by *simp-all*
  then have ‹$\forall\, t \in \mathcal{K}\ M\ i\ s.\ M,\ t \models p$› ‹$\forall\, t \in \mathcal{K}\ M\ i\ s.\ M,\ t \models (p \longrightarrow q)$›
    by *simp-all*
  then have ‹$\forall\, t \in \mathcal{K}\ M\ i\ s.\ M,\ t \models q$›
    by *simp*
  then show ‹M, $s \models K\ i\ q$›
    by *simp*
qed

theorem *generalization*:
  assumes *valid*: ‹$\forall\, (M :: ('i, 's)\ kripke)\ s.\ M,\ s \models p$›
  shows ‹$(M :: ('i, 's)\ kripke), s \models K\ i\ p$›
proof −
  have ‹$\forall\, t \in \mathcal{K}\ M\ i\ s.\ M,\ t \models p$›
    using *valid* **by** *blast*
  then show ‹M, $s \models K\ i\ p$›
    by *simp*
qed

theorem *truth*:
  assumes ‹*reflexive M*›
  shows ‹M, $s \models (K\ i\ p \longrightarrow p)$›
proof (*rule Imp-intro*)
  assume ‹M, $s \models K\ i\ p$›
  then have ‹$\forall\, t \in \mathcal{K}\ M\ i\ s.\ M,\ t \models p$›
    by *simp*
  moreover have ‹$s \in \mathcal{K}\ M\ i\ s$›
    using ‹*reflexive M*› **by** *blast*
  ultimately show ‹M, $s \models p$›
    by *blast*
qed

theorem *pos-introspection*:
  assumes ‹*transitive M*›
  shows ‹M, $s \models (K\ i\ p \longrightarrow K\ i\ (K\ i\ p))$›
proof (*rule Imp-intro*)
  assume ‹M,$s \models K\ i\ p$›
  then have ‹$\forall\, t \in \mathcal{K}\ M\ i\ s.\ M,\ t \models p$›
    by *simp*
  then have ‹$\forall\, t \in \mathcal{K}\ M\ i\ s.\ \forall\, u \in \mathcal{K}\ M\ i\ t.\ M,\ u \models p$›
    using ‹*transitive M*› **by** *blast*
  then have ‹$\forall\, t \in \mathcal{K}\ M\ i\ s.\ M,\ t \models K\ i\ p$›
    by *simp*

then show ‹M, $s \models K\ i\ (K\ i\ p)$›
by *simp*
qed

theorem *neg-introspection*:
assumes ‹*symmetric M*› ‹*transitive M*›
shows ‹M, $s \models (\neg\ K\ i\ p \longrightarrow K\ i\ (\neg\ K\ i\ p))$›
proof (*rule Imp-intro*)
assume ‹M,$s \models \neg\ (K\ i\ p)$›
then obtain u **where** ‹$u \in \mathcal{K}\ M\ i\ s$› ‹$\neg\ (M,\ u \models p)$›
by *auto*
moreover have ‹$\forall\ t \in \mathcal{K}\ M\ i\ s.\ u \in \mathcal{K}\ M\ i\ t$›
using ‹$u \in \mathcal{K}\ M\ i\ s$› ‹*symmetric M*› ‹*transitive M*› **by** *blast*
ultimately have ‹$\forall\ t \in \mathcal{K}\ M\ i\ s.\ M,\ t \models \neg\ K\ i\ p$›
by *auto*
then show ‹M, $s \models K\ i\ (\neg\ K\ i\ p)$›
by *simp*
qed

5 Axiom System K

primrec *eval* :: ‹(*id* ⇒ *bool*) ⇒ (′*i fm* ⇒ *bool*) ⇒ ′*i fm* ⇒ *bool*› **where**
‹*eval* - - ⊥ = *False*›
| ‹*eval g* - (*Pro i*) = *g i*›
| ‹*eval g h* ($p \vee q$) = (*eval g h p* ∨ *eval g h q*)›
| ‹*eval g h* ($p \wedge q$) = (*eval g h p* ∧ *eval g h q*)›
| ‹*eval g h* ($p \longrightarrow q$) = (*eval g h p* ⟶ *eval g h q*)›
| ‹*eval* - *h* (*K i p*) = *h* (*K i p*)›

abbreviation ‹*tautology p* ≡ ∀ *g h*. *eval g h p*›

inductive *SystemK* :: ‹′*i fm* ⇒ *bool*› (⊢ - [*50*] *50*) **where**
A1: ‹*tautology p* ⟹ ⊢ *p*›
| *A2*: ‹⊢ (*K i p* ∧ *K i* ($p \longrightarrow q$) ⟶ *K i q*)›
| *R1*: ‹⊢ *p* ⟹ ⊢ ($p \longrightarrow q$) ⟹ ⊢ *q*›
| *R2*: ‹⊢ *p* ⟹ ⊢ *K i p*›

6 Soundness

lemma *eval-semantics*: ‹*eval* (*pi s*) (λ*q*. *Kripke pi r*, *s* ⊨ *q*) *p* = (*Kripke pi r*, *s* ⊨ *p*)›
by (*induct p*) *simp-all*

theorem *tautology*: ‹*tautology p* ⟹ *M*, *s* ⊨ *p*›
proof −
assume ‹*tautology p*›
then have ‹*eval* (*g s*) (λ*q*. *Kripke g r*, *s* ⊨ *q*) *p*› **for** *g r*
by *simp*

then have ‹*Kripke g r*, $s \models p$› **for** *g r*
using *eval-semantics* **by** *metis*
then show ‹M, $s \models p$›
by (*metis kripke.collapse*)
qed

theorem *soundness*: ‹$\vdash p \Longrightarrow M, s \models p$›
by (*induct p arbitrary*: *s rule*: *SystemK.induct*) (*simp-all add*: *tautology*)

7 Derived rules

lemma *K-FFI*: ‹$\vdash (p \longrightarrow (\neg\ p) \longrightarrow \bot)$›
by (*simp add*: *A1*)

primrec *conjoin* :: ‹*'i fm list* ⇒ *'i fm* ⇒ *'i fm*› **where**
‹*conjoin* [] *q* = *q*›
| ‹*conjoin* (*p* # *ps*) *q* = (*p* ∧ *conjoin ps q*)›

primrec *imply* :: ‹*'i fm list* ⇒ *'i fm* ⇒ *'i fm*› **where**
‹*imply* [] *q* = *q*›
| ‹*imply* (*p* # *ps*) *q* = (*p* ⟶ *imply ps q*)›

lemma *K-imply-head*: ‹$\vdash$ *imply* (*p* # *ps*) *p*›
proof −
have ‹*tautology* (*imply* (*p* # *ps*) *p*)›
by (*induct ps*) *simp-all*
then show *?thesis*
using *A1* **by** *blast*
qed

lemma *K-imply-Cons*:
assumes ‹$\vdash$ *imply ps q*›
shows ‹$\vdash$ *imply* (*p* # *ps*) *q*›
proof −
have ‹*tautology* (*imply ps q* ⟶ *imply* (*p* # *ps*) *q*)›
by *simp*
then have ‹$\vdash$ (*imply ps q* ⟶ *imply* (*p* # *ps*) *q*)›
using *A1* **by** *blast*
then show *?thesis*
using *assms R1* **by** *blast*
qed

lemma *K-imply-member*: ‹*p* ∈ *set ps* ⟹ $\vdash$ *imply ps p*›
proof (*induct ps*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons a ps*)

```
  then show ?case
  proof (cases ‹a = p›)
    case True
    then show ?thesis
      using Cons K-imply-head by blast
  next
    case False
    then show ?thesis
      using Cons K-imply-Cons by simp
  qed
qed

lemma K-right-mp:
  assumes ‹⊢ imply ps p› ‹⊢ imply ps (p ⟶ q)›
  shows ‹⊢ imply ps q›
proof −
  have ‹tautology (imply ps p ⟶ imply ps (p ⟶ q) ⟶ imply ps q)›
    by (induct ps) simp-all
  then have ‹⊢ (imply ps p ⟶ imply ps (p ⟶ q) ⟶ imply ps q)›
    using A1 by blast
  then show ?thesis
    using assms R1 by blast
qed

lemma tautology-imply-superset:
  assumes ‹set ps ⊆ set qs›
  shows ‹tautology (imply ps r ⟶ imply qs r)›
proof (rule ccontr)
  assume ‹¬ tautology (imply ps r ⟶ imply qs r)›
  then obtain g h where ‹¬ eval g h (imply ps r ⟶ imply qs r)›
    by blast
  then have ‹eval g h (imply ps r)› ‹¬ eval g h (imply qs r)›
    by simp-all
  then consider (np) ‹∃ p ∈ set ps. ¬ eval g h p› | (r) ‹∀ p ∈ set ps. eval g h p›
‹eval g h r›
    by (induct ps) auto
  then show False
  proof cases
    case np
    then have ‹∃ p ∈ set qs. ¬ eval g h p›
      using ‹set ps ⊆ set qs› by blast
    then have ‹eval g h (imply qs r)›
      by (induct qs) simp-all
    then show ?thesis
      using ‹¬ eval g h (imply qs r)› by blast
  next
    case r
    then have ‹eval g h (imply qs r)›
      by (induct qs) simp-all
```

```
    then show ?thesis
      using ‹¬ eval g h (imply qs r)› by blast
  qed
qed

lemma tautology-imply: ‹tautology q ⟹ tautology (imply ps q)›
  by (induct ps) simp-all

theorem K-imply-weaken:
  assumes ‹⊢ imply ps q› ‹set ps ⊆ set ps'›
  shows ‹⊢ imply ps' q›
proof −
  have ‹tautology (imply ps q ⟶ imply ps' q)›
    using ‹set ps ⊆ set ps'›
  proof (induct ps arbitrary: ps')
    case Nil
    then show ?case
      by (induct ps') simp-all
  next
    case (Cons a G)
    then show ?case
      using tautology-imply-superset by blast
  qed
  then have ‹⊢ (imply ps q ⟶ imply ps' q)›
    using A1 by blast
  then show ?thesis
    using ‹⊢ imply ps q› R1 by blast
qed

lemma imply-append: ‹imply (ps @ ps') q = imply ps (imply ps' q)›
  by (induct ps) simp-all

lemma K-ImpI:
  assumes ‹⊢ imply (p # G) q›
  shows ‹⊢ imply G (p ⟶ q)›
proof −
  have ‹set (p # G) ⊆ set (G @ [p])›
    by simp
  then have ‹⊢ imply (G @ [p]) q›
    using assms K-imply-weaken by blast
  then have ‹⊢ imply G (imply [p] q)›
    using imply-append by metis
  then show ?thesis
    by simp
qed

lemma cut: ‹⊢ imply G p ⟹ ⊢ imply (p # G) q ⟹ ⊢ imply G q›
  using K-ImpI K-right-mp by blast
```

lemma *K-Boole*: ‹⊢ *imply* ((¬ *p*) # *G*) ⊥ ⟹ ⊢ *imply G p*›
proof −
  assume ‹⊢ *imply* ((¬ *p*) # *G*) ⊥›
  then have ‹⊢ *imply G* (¬ ¬ *p*)›
    using *K-ImpI* **by** *blast*
  moreover have ‹*tautology* (*imply G* (¬ ¬ *p*) ⟶ *imply G p*)›
    by (*induct G*) *simp-all*
  then have ‹⊢ (*imply G* (¬ ¬ *p*) ⟶ *imply G p*)›
    using *A1* **by** *blast*
  ultimately show *?thesis*
    using *R1* **by** *blast*
qed

lemma *K-DisE*:
  assumes ‹⊢ *imply* (*A* # *G*) *C*› ‹⊢ *imply* (*B* # *G*) *C*› ‹⊢ *imply G* (*A* ∨ *B*)›
  shows ‹⊢ *imply G C*›
proof −
  have ‹*tautology* (*imply* (*A* # *G*) *C* ⟶ *imply* (*B* # *G*) *C* ⟶ *imply G* (*A* ∨ *B*) ⟶ *imply G C*)›
    by (*induct G*) *auto*
  then have ‹⊢ (*imply* (*A* # *G*) *C* ⟶ *imply* (*B* # *G*) *C* ⟶ *imply G* (*A* ∨ *B*) ⟶ *imply G C*)›
    using *A1* **by** *blast*
  then show *?thesis*
    using *assms R1* **by** *blast*
qed

lemma *K-conjoin-imply*:
  assumes ‹⊢ (¬ *conjoin G* (¬ *p*))›
  shows ‹⊢ *imply G p*›
proof −
  have ‹*tautology* (¬ *conjoin G* (¬ *p*) ⟶ *imply G p*)›
    by (*induct G*) *simp-all*
  then have ‹⊢ (¬ *conjoin G* (¬ *p*) ⟶ *imply G p*)›
    using *A1* **by** *blast*
  then show *?thesis*
    using *assms R1* **by** *blast*
qed

lemma *K-distrib-K-imp*:
  assumes ‹⊢ *K i* (*imply G q*)›
  shows ‹⊢ *imply* (*map* (*K i*) *G*) (*K i q*)›
proof −
  have ‹⊢ (*K i* (*imply G q*) ⟶ *imply* (*map* (*K i*) *G*) (*K i q*))›
  proof (*induct G*)
    case *Nil*
    then show *?case*
      by (*simp add*: *A1*)
  next

case (*Cons a G*)
have ‹$\vdash (K\ i\ a \wedge K\ i\ (imply\ (a\ \#\ G)\ q) \longrightarrow K\ i\ (imply\ G\ q))$›
by (*simp add*: *A2*)
moreover have
‹$\vdash ((K\ i\ a \wedge K\ i\ (imply\ (a\ \#\ G)\ q) \longrightarrow K\ i\ (imply\ G\ q)) \longrightarrow$
$(K\ i\ (imply\ G\ q) \longrightarrow imply\ (map\ (K\ i)\ G)\ (K\ i\ q)) \longrightarrow$
$(K\ i\ a \wedge K\ i\ (imply\ (a\ \#\ G)\ q) \longrightarrow imply\ (map\ (K\ i)\ G)\ (K\ i\ q)))$›
by (*simp add*: *A1*)
ultimately have ‹$\vdash (K\ i\ a \wedge K\ i\ (imply\ (a\ \#\ G)\ q) \longrightarrow imply\ (map\ (K\ i)\ G)\ (K\ i\ q))$›
using *Cons R1* **by** *blast*
moreover have
‹$\vdash ((K\ i\ a \wedge K\ i\ (imply\ (a\ \#\ G)\ q) \longrightarrow imply\ (map\ (K\ i)\ G)\ (K\ i\ q)) \longrightarrow$
$(K\ i\ (imply\ (a\ \#\ G)\ q) \longrightarrow K\ i\ a \longrightarrow imply\ (map\ (K\ i)\ G)\ (K\ i\ q)))$›
by (*simp add*: *A1*)
ultimately have ‹$\vdash (K\ i\ (imply\ (a\ \#\ G)\ q) \longrightarrow K\ i\ a \longrightarrow imply\ (map\ (K\ i)\ G)\ (K\ i\ q))$›
using *R1* **by** *blast*
then show *?case*
by *simp*
qed
then show *?thesis*
using *assms R1* **by** *blast*
qed

8 Consistency

definition *consistency* :: ‹*'i fm set set* ⇒ *bool*› **where**
‹*consistency C* ≡ ∀ *S* ∈ *C*.
(∀ *p*. ¬ (*Pro p* ∈ *S* ∧ (¬ *Pro p*) ∈ *S*)) ∧
⊥ ∉ *S* ∧
(∀ *Z*. (¬ (¬ *Z*)) ∈ *S* ⟶ *S* ∪ {*Z*} ∈ *C*) ∧
(∀ *A B*. (*A* ∧ *B*) ∈ *S* ⟶ *S* ∪ {*A*, *B*} ∈ *C*) ∧
(∀ *A B*. (¬ (*A* ∨ *B*)) ∈ *S* ⟶ *S* ∪ {¬ *A*, ¬ *B*} ∈ *C*) ∧
(∀ *A B*. (*A* ∨ *B*) ∈ *S* ⟶ *S* ∪ {*A*} ∈ *C* ∨ *S* ∪ {*B*} ∈ *C*) ∧
(∀ *A B*. (¬ (*A* ∧ *B*)) ∈ *S* ⟶ *S* ∪ {¬ *A*} ∈ *C* ∨ *S* ∪ {¬ *B*} ∈ *C*) ∧
(∀ *A B*. (*A* ⟶ *B*) ∈ *S* ⟶ *S* ∪ {¬ *A*} ∈ *C* ∨ *S* ∪ {*B*} ∈ *C*) ∧
(∀ *A B*. (¬ (*A* ⟶ *B*)) ∈ *S* ⟶ *S* ∪ {*A*, ¬ *B*} ∈ *C*) ∧
(∀ *A*. *tautology A* ⟶ *S* ∪ {*A*} ∈ *C*) ∧
(∀ *A i*. ¬ (*K i A* ∈ *S* ∧ (¬ *K i A*) ∈ *S*))›

8.1 Closure under subsets

definition *close* :: ‹*'i fm set set* ⇒ *'i fm set set*› **where**
‹*close C* ≡ {*S*. ∃ *S'* ∈ *C*. *S* ⊆ *S'*}›

definition *subset-closed* :: ‹*'a set set* ⇒ *bool*› **where**
‹*subset-closed C* ≡ (∀ *S'* ∈ *C*. ∀ *S*. *S* ⊆ *S'* ⟶ *S* ∈ *C*)›

lemma *subset-in-close*:
assumes ‹$S' \subseteq S$› ‹$S \cup x \in C$›
shows ‹$S' \cup x \in$ *close* C›
proof −
have ‹$S \cup x \in$ *close* C›
unfolding *close-def* **using** ‹$S \cup x \in C$› **by** *blast*
then show *?thesis*
unfolding *close-def* **using** ‹$S' \subseteq S$› **by** *blast*
qed

theorem *close-consistency*:
fixes C :: ‹*'i fm set set*›
assumes ‹*consistency* C›
shows ‹*consistency* (*close* C)›
unfolding *consistency-def*
proof (*intro ballI allI impI conjI*)
fix S' :: ‹*'i fm set*›
assume ‹$S' \in$ *close* C›
then obtain S **where** ‹$S \in C$› ‹$S' \subseteq S$›
unfolding *close-def* **by** *blast*

{ fix p
have ‹$\neg$ (*Pro* $p \in S \wedge (\neg$ *Pro* $p) \in S$)›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$\neg$ (*Pro* $p \in S' \wedge (\neg$ *Pro* $p) \in S'$)›
using ‹$S' \subseteq S$› **by** *blast* **}**

{ have ‹$\bot \notin S$›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *blast*
then show ‹$\bot \notin S'$›
using ‹$S' \subseteq S$› **by** *blast* **}**

{ fix Z
assume ‹$(\neg (\neg Z)) \in S'$›
then have ‹$(\neg (\neg Z)) \in S$›
using ‹$S' \subseteq S$› **by** *blast*
then have ‹$S \cup \{Z\} \in C$›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$S' \cup \{Z\} \in$ *close* C›
using ‹$S' \subseteq S$› *subset-in-close* **by** *blast* **}**

{ fix A B
assume ‹$(A \wedge B) \in S'$›
then have ‹$(A \wedge B) \in S$›
using ‹$S' \subseteq S$› **by** *blast*
then have ‹$S \cup \{A, B\} \in C$›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$S' \cup \{A, B\} \in$ *close* C›
using ‹$S' \subseteq S$› *subset-in-close* **by** *blast* **}**

{ fix A B
assume ‹$(\neg\ (A \vee B)) \in S'$›
then have ‹$(\neg\ (A \vee B)) \in S$›
using ‹$S' \subseteq S$› **by** *blast*
then have ‹$S \cup \{\neg\ A, \neg\ B\} \in C$›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$S' \cup \{\neg\ A, \neg\ B\} \in$ *close* C›
using ‹$S' \subseteq S$› *subset-in-close* **by** *blast* **}**

{ fix A B
assume ‹$(\neg\ (A \longrightarrow B)) \in S'$›
then have ‹$(\neg\ (A \longrightarrow B)) \in S$›
using ‹$S' \subseteq S$› **by** *blast*
then have ‹$S \cup \{A, \neg\ B\} \in C$›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *metis*
then show ‹$S' \cup \{A, \neg\ B\} \in$ *close* C›
using ‹$S' \subseteq S$› *subset-in-close* **by** *blast* **}**

{ fix A B
assume ‹$(A \vee B) \in S'$›
then have ‹$(A \vee B) \in S$›
using ‹$S' \subseteq S$› **by** *blast*
then have ‹$S \cup \{A\} \in C \vee S \cup \{B\} \in C$›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$S' \cup \{A\} \in$ *close* $C \vee S' \cup \{B\} \in$ *close* C›
using ‹$S' \subseteq S$› *subset-in-close* **by** *blast* **}**

{ fix A B
assume ‹$(\neg\ (A \wedge B)) \in S'$›
then have ‹$(\neg\ (A \wedge B)) \in S$›
using ‹$S' \subseteq S$› **by** *blast*
then have ‹$S \cup \{\neg\ A\} \in C \vee S \cup \{\neg\ B\} \in C$›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$S' \cup \{\neg\ A\} \in$ *close* $C \vee S' \cup \{\neg\ B\} \in$ *close* C›
using ‹$S' \subseteq S$› *subset-in-close* **by** *blast* **}**

{ fix A B
assume ‹$(A \longrightarrow B) \in S'$›
then have ‹$(A \longrightarrow B) \in S$›
using ‹$S' \subseteq S$› **by** *blast*
then have ‹$S \cup \{\neg\ A\} \in C \vee S \cup \{B\} \in C$›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$S' \cup \{\neg\ A\} \in$ *close* $C \vee S' \cup \{B\} \in$ *close* C›
using ‹$S' \subseteq S$› *subset-in-close* **by** *blast* **}**

{ fix A :: ‹*'i fm*›
assume ‹*tautology* A›
then have ‹$S \cup \{A\} \in C$›

using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$S' \cup \{A\} \in$ *close* C›
using ‹$S' \subseteq S$› *subset-in-close* **by** *blast* **}**

{ fix A i
have ‹$\neg\ (K\ i\ A \in S \wedge (\neg\ K\ i\ A) \in S)$›
using ‹$S \in C$› ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$\neg\ (K\ i\ A \in S' \wedge (\neg\ K\ i\ A) \in S')$›
using ‹$S' \subseteq S$› **by** *blast* **}**

qed

theorem *close-closed*: ‹*subset-closed* (*close* C)›
unfolding *close-def subset-closed-def* **by** *blast*

theorem *close-subset*: ‹$C \subseteq$ *close* C›
unfolding *close-def* **by** *blast*

8.2 Finite character

definition *finite-char* :: ‹$'a$ *set set* $\Rightarrow$ *bool*› **where**
‹*finite-char* $C \equiv (\forall S.\ S \in C = (\forall S'.$ *finite* $S' \longrightarrow S' \subseteq S \longrightarrow S' \in C))$›

definition *mk-finite-char* :: ‹$'a$ *set set* $\Rightarrow$ $'a$ *set set*› **where**
‹*mk-finite-char* $C \equiv \{S.\ \forall S'.$ *finite* $S' \longrightarrow S' \subseteq S \longrightarrow S' \in C\}$›

theorem *finite-char*: ‹*finite-char* (*mk-finite-char* C)›
unfolding *finite-char-def mk-finite-char-def* **by** *blast*

theorem *finite-char-closed*: ‹*finite-char* $C \Longrightarrow$ *subset-closed* C›
unfolding *finite-char-def subset-closed-def*
proof (*intro ballI allI impI*)
fix S S'
assume $*$: ‹$\forall S.\ (S \in C) = (\forall S'.$ *finite* $S' \longrightarrow S' \subseteq S \longrightarrow S' \in C)$›
and ‹$S' \in C$› ‹$S \subseteq S'$›
then have ‹$\forall S'.$ *finite* $S' \longrightarrow S' \subseteq S \longrightarrow S' \in C$›
by *blast*
then show ‹$S \in C$›
using $*$ **by** *blast*
qed

theorem *finite-char-subset*: ‹*subset-closed* $C \Longrightarrow C \subseteq$ *mk-finite-char* C›
unfolding *mk-finite-char-def subset-closed-def* **by** *blast*

theorem *finite-consistency*:
fixes C :: ‹$'i$ *fm set set*›
assumes ‹*consistency* C› ‹*subset-closed* C›
shows ‹*consistency* (*mk-finite-char* C)›
unfolding *consistency-def*

```
proof (intro ballI allI impI conjI)
  fix S
  assume ‹S ∈ mk-finite-char C›
  then have finc: ‹∀ S' ⊆ S. finite S' ⟶ S' ∈ C›
    unfolding mk-finite-char-def by blast

  have ‹∀ S' ∈ C. ∀ S ⊆ S'. S ∈ C›
    using ‹subset-closed C› unfolding subset-closed-def by blast
  then have sc: ‹∀ S' x. S' ∪ x ∈ C ⟶ (∀ S ⊆ S' ∪ x. S ∈ C)›
    by blast

  { fix i
    show ‹¬ (Pro i ∈ S ∧ (¬ Pro i) ∈ S)›
    proof
      assume ‹Pro i ∈ S ∧ (¬ Pro i) ∈ S›
      then have ‹{Pro i, (¬ Pro i)} ∈ C›
        using finc by simp
      then show False
        using ‹consistency C› unfolding consistency-def by fast
    qed }

  show ‹⊥ ∉ S›
  proof
    assume ‹⊥ ∈ S›
    then have ‹{⊥} ∈ C›
      using finc by simp
    then show False
      using ‹consistency C› unfolding consistency-def by fast
  qed

  { fix Z
    assume ∗: ‹(¬ ¬ Z) ∈ S›
    show ‹S ∪ {Z} ∈ mk-finite-char C›
      unfolding mk-finite-char-def
    proof (intro allI impI CollectI)
      fix S'
      let ?S' = ‹S' − {Z} ∪ {¬ ¬ Z}›

      assume ‹S' ⊆ S ∪ {Z}› ‹finite S'›
      then have ‹?S' ⊆ S›
        using ∗ by blast
      moreover have ‹finite ?S'›
        using ‹finite S'› by blast
      ultimately have ‹?S' ∈ C›
        using finc by blast
      then have ‹?S' ∪ {Z} ∈ C›
        using ‹consistency C› unfolding consistency-def by simp
      then show ‹S' ∈ C›
        using sc by blast
```

qed }

{ **fix** A B
assume $*$: ‹$(A \wedge B) \in S$›
show ‹$S \cup \{A, B\} \in$ *mk-finite-char* C›
unfolding *mk-finite-char-def*
proof (*intro allI impI CollectI*)
fix S'
let *?S'* = ‹$S' - \{A, B\} \cup \{A \wedge B\}$›

assume ‹$S' \subseteq S \cup \{A, B\}$› ‹*finite* S'›
then have ‹*?S'* $\subseteq S$›
using $*$ **by** *blast*
moreover have ‹*finite ?S'*›
using ‹*finite* S'› **by** *blast*
ultimately have ‹*?S'* $\in C$›
using *finc* **by** *blast*
then have ‹*?S'* $\cup \{A, B\} \in C$›
using ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$S' \in C$›
using *sc* **by** *blast*
qed }

{ **fix** A B
assume $*$: ‹$(\neg (A \vee B)) \in S$›
show ‹$S \cup \{\neg A, \neg B\} \in$ *mk-finite-char* C›
unfolding *mk-finite-char-def*
proof (*intro allI impI CollectI*)
fix S'
let *?S'* = ‹$S' - \{\neg A, \neg B\} \cup \{\neg (A \vee B)\}$›

assume ‹$S' \subseteq S \cup \{\neg A, \neg B\}$› ‹*finite* S'›
then have ‹*?S'* $\subseteq S$›
using $*$ **by** *blast*
moreover have ‹*finite ?S'*›
using ‹*finite* S'› **by** *blast*
ultimately have ‹*?S'* $\in C$›
using *finc* **by** *blast*
then have ‹*?S'* $\cup \{\neg A, \neg B\} \in C$›
using ‹*consistency* C› **unfolding** *consistency-def* **by** *simp*
then show ‹$S' \in C$›
using *sc* **by** *blast*
qed }

{ **fix** A B
assume $*$: ‹$(\neg (A \longrightarrow B)) \in S$›
show ‹$S \cup \{A, \neg B\} \in$ *mk-finite-char* C›
unfolding *mk-finite-char-def*
proof (*intro allI impI CollectI*)

```
    fix S'
    let ?S' = ‹S' − {A, ¬ B} ∪ {¬ (A ⟶ B)}›

    assume ‹S' ⊆ S ∪ {A, ¬ B}› ‹finite S'›
    then have ‹?S' ⊆ S›
      using ∗ by blast
    moreover have ‹finite ?S'›
      using ‹finite S'› by blast
    ultimately have ‹?S' ∈ C›
      using finc by blast
    then have ‹?S' ∪ {A, ¬ B} ∈ C›
      using ‹consistency C› unfolding consistency-def by simp
    then show ‹S' ∈ C›
      using sc by blast
  qed }

{ fix A B
  assume ∗: ‹(A ∨ B) ∈ S›
  show ‹S ∪ {A} ∈ mk-finite-char C ∨ S ∪ {B} ∈ mk-finite-char C›
  proof (rule ccontr)
    assume ‹¬ ?thesis›
    then obtain Sa Sb
      where ‹Sa ⊆ S ∪ {A}› ‹finite Sa› ‹Sa ∉ C›
        and ‹Sb ⊆ S ∪ {B}› ‹finite Sb› ‹Sb ∉ C›
      unfolding mk-finite-char-def by blast

    let ?S' = ‹(Sa − {A}) ∪ (Sb − {B}) ∪ {A ∨ B}›

    have ‹?S' ⊆ S›
      using ‹Sa ⊆ S ∪ {A}› ‹Sb ⊆ S ∪ {B}› ∗ by blast
    moreover have ‹finite ?S'›
      using ‹finite Sa› ‹finite Sb› by blast
    ultimately have ‹?S' ∈ C›
      using finc by blast
    then have ‹?S' ∪ {A} ∈ C ∨ ?S' ∪ {B} ∈ C›
      using ‹consistency C› unfolding consistency-def by simp
    then have ‹Sa ∈ C ∨ Sb ∈ C›
      using sc by blast
    then show False
      using ‹Sa ∉ C› ‹Sb ∉ C› by blast
  qed }

{ fix A B
  assume ∗: ‹(¬ (A ∧ B)) ∈ S›
  show ‹S ∪ {¬ A} ∈ mk-finite-char C ∨ S ∪ {¬ B} ∈ mk-finite-char C›
  proof (rule ccontr)
    assume ‹¬ ?thesis›
    then obtain Sa Sb
      where ‹Sa ⊆ S ∪ {¬ A}› ‹finite Sa› ‹Sa ∉ C›
```

```
      and ‹Sb ⊆ S ∪ {¬ B}› ‹finite Sb› ‹Sb ∉ C›
    unfolding mk-finite-char-def by blast

  let ?S' = ‹(Sa − {¬ A}) ∪ (Sb − {¬ B}) ∪ {¬ (A ∧ B)}›

  have ‹?S' ⊆ S›
    using ‹Sa ⊆ S ∪ {¬ A}› ‹Sb ⊆ S ∪ {¬ B}› ∗ by blast
  moreover have ‹finite ?S'›
    using ‹finite Sa› ‹finite Sb› by blast
  ultimately have ‹?S' ∈ C›
    using finc by blast
  then have ‹?S' ∪ {¬ A} ∈ C ∨ ?S' ∪ {¬ B} ∈ C›
    using ‹consistency C› unfolding consistency-def by simp
  then have ‹Sa ∈ C ∨ Sb ∈ C›
    using sc by blast
  then show False
    using ‹Sa ∉ C› ‹Sb ∉ C› by blast
 qed }

{ fix A B
 assume ∗: ‹(A ⟶ B) ∈ S›
 show ‹S ∪ {¬ A} ∈ mk-finite-char C ∨ S ∪ {B} ∈ mk-finite-char C›
 proof (rule ccontr)
   assume ‹¬ ?thesis›
   then obtain Sa Sb
     where ‹Sa ⊆ S ∪ {¬ A}› ‹finite Sa› ‹Sa ∉ C›
       and ‹Sb ⊆ S ∪ {B}› ‹finite Sb› ‹Sb ∉ C›
     unfolding mk-finite-char-def by blast

   let ?S' = ‹(Sa − {¬ A}) ∪ (Sb − {B}) ∪ {A ⟶ B}›

   have ‹?S' ⊆ S›
     using ‹Sa ⊆ S ∪ {¬ A}› ‹Sb ⊆ S ∪ {B}› ∗ by blast
   moreover have ‹finite ?S'›
     using ‹finite Sa› ‹finite Sb› by blast
   ultimately have ‹?S' ∈ C›
     using finc by blast
   then have ‹?S' ∪ {¬ A} ∈ C ∨ ?S' ∪ {B} ∈ C›
     using ‹consistency C› unfolding consistency-def by simp
   then have ‹Sa ∈ C ∨ Sb ∈ C›
     using sc by blast
   then show False
     using ‹Sa ∉ C› ‹Sb ∉ C› by blast
 qed }

{ fix A :: ‹'i fm›
 assume ∗: ‹tautology A›
 show ‹S ∪ {A} ∈ mk-finite-char C›
   unfolding mk-finite-char-def
```

proof (*intro allI impI CollectI*)
fix S'
let $?S' = \langle S' - \{A\}\rangle$

assume $\langle S' \subseteq S \cup \{A\}\rangle$ $\langle$*finite* $S'\rangle$
then have $\langle ?S' \subseteq S\rangle$
using $*$ **by** *blast*
moreover have $\langle$*finite* $?S'\rangle$
using $\langle$*finite* $S'\rangle$ **by** *blast*
ultimately have $\langle ?S' \in C\rangle$
using *finc* **by** *blast*
then have $\langle ?S' \cup \{A\} \in C\rangle$
using $*$ $\langle$*consistency* $C\rangle$ **unfolding** *consistency-def* **by** *metis*
then show $\langle S' \in C\rangle$
using *sc* **by** *blast*
qed }

{ **fix** A i
assume $\langle S \in$ *mk-finite-char* $C\rangle$
show $\langle \neg\ (K\ i\ A \in S \wedge (\neg\ K\ i\ A) \in S)\rangle$
proof
assume $\langle K\ i\ A \in S \wedge (\neg\ K\ i\ A) \in S\rangle$
then have $\langle \{K\ i\ A, (\neg\ K\ i\ A)\} \in C\rangle$
using *finc* **by** *simp*
then show *False*
using $\langle$*consistency* $C\rangle$ **unfolding** *consistency-def* **by** *auto*
qed }

qed

8.3 Maximal extension

instantiation *fm* :: (*countable*) *countable* **begin**
instance by *countable-datatype*
end

definition *is-chain* :: $\langle(nat \Rightarrow 'a\ set) \Rightarrow bool\rangle$ **where**
$\langle$*is-chain* $f \equiv \forall\, n.\ f\ n \subseteq f\ (Suc\ n)\rangle$

lemma *is-chainD*: $\langle$*is-chain* $f \Longrightarrow x \in f\ m \Longrightarrow x \in f\ (m + n)\rangle$
by (*induct n*) (*auto simp*: *is-chain-def*)

lemma *is-chainD'*:
assumes $\langle$*is-chain* $f\rangle$ $\langle x \in f\ m\rangle$ $\langle m \leq k\rangle$
shows $\langle x \in f\ k\rangle$
proof −
obtain n **where** $\langle k = m + n\rangle$
using $\langle m \leq k\rangle$ *le-iff-add*
by *metis*

then show ‹$x \in f\ k$›
using ‹*is-chain f*› ‹$x \in f\ m$› *is-chainD*
by *metis*
qed

lemma *chain-index*:
assumes ‹*is-chain f*› ‹*finite F*›
shows ‹$F \subseteq (\bigcup n.\ f\ n) \Longrightarrow \exists n.\ F \subseteq f\ n$›
using ‹*finite F*›
proof (*induct F rule*: *finite-induct*)
case *empty*
then show *?case*
by *blast*
next
case (*insert x F*)
then have ‹$\exists n.\ F \subseteq f\ n$› ‹$\exists m.\ x \in f\ m$› ‹$F \subseteq (\bigcup x.\ f\ x)$›
using ‹*is-chain f*› **by** *simp-all*
then obtain n **and** m **where** ‹$F \subseteq f\ n$› ‹$x \in f\ m$›
by *blast*
have ‹$m \leq max\ n\ m$› ‹$n \leq max\ n\ m$›
by *simp-all*
have ‹$x \in f\ (max\ n\ m)$›
using *is-chainD'* ‹*is-chain f*› ‹$x \in f\ m$› ‹$m \leq max\ n\ m$›
by *fast*
moreover have ‹$F \subseteq f\ (max\ n\ m)$›
using *is-chainD'* ‹*is-chain f*› ‹$F \subseteq f\ n$› ‹$n \leq max\ n\ m$›
by *fast*
ultimately show *?case*
by *blast*
qed

lemma *chain-union-closed'*:
assumes ‹*is-chain f*› ‹$\forall n.\ f\ n \in C$› ‹$\forall S' \in C.\ \forall S \subseteq S'.\ S \in C$› ‹*finite S'*› ‹$S' \subseteq (\bigcup n.\ f\ n)$›
shows ‹$S' \in C$›
proof −
obtain n **where** ‹$S' \subseteq f\ n$›
using *chain-index* ‹*is-chain f*› ‹*finite S'*› ‹$S' \subseteq (\bigcup n.\ f\ n)$›
by *blast*
moreover have ‹$f\ n \in C$›
using ‹$\forall n.\ f\ n \in C$›
by *blast*
ultimately show ‹$S' \in C$›
using ‹$\forall S' \in C.\ \forall S \subseteq S'.\ S \in C$›
by *blast*
qed

lemma *chain-union-closed*:
assumes ‹*finite-char C*› ‹*is-chain f*› ‹$\forall n.\ f\ n \in C$›

shows ‹$(\bigcup n.\ f\ n) \in C$›
proof −
have ‹*subset-closed C*›
using *finite-char-closed* ‹*finite-char C*›
by *blast*
then have ‹$\forall S' \in C.\ \forall S \subseteq S'.\ S \in C$›
unfolding *subset-closed-def*
by *blast*
then have ‹$\forall S'.$ *finite* $S' \longrightarrow S' \subseteq (\bigcup n.\ f\ n) \longrightarrow S' \in C$›
using *chain-union-closed′ assms*
by *blast*
moreover have ‹$((\bigcup n.\ f\ n) \in C) = (\forall S'.$ *finite* $S' \longrightarrow S' \subseteq (\bigcup n.\ f\ n) \longrightarrow S' \in C)$›
using ‹*finite-char C*› **unfolding** *finite-char-def*
by *blast*
ultimately show *?thesis*
by *blast*
qed

primrec *extend* :: ‹*'i fm set* ⇒ *'i fm set set* ⇒ (*nat* ⇒ *'i fm*) ⇒ *nat* ⇒ *'i fm set*›
where
‹*extend S C f 0* = *S*› |
‹*extend S C f* (*Suc n*) =
(*if extend S C f n* ∪ {*f n*} ∈ *C*
then extend S C f n ∪ {*f n*}
else extend S C f n)›

definition *Extend* :: ‹*'i fm set* ⇒ *'i fm set set* ⇒ (*nat* ⇒ *'i fm*) ⇒ *'i fm set*› **where**
‹*Extend S C f* ≡ ⋃ *n. extend S C f n*›

lemma *is-chain-extend*: ‹*is-chain* (*extend S C f*)›
by (*simp add*: *is-chain-def*) *blast*

lemma *extend-in-C*: ‹*consistency C* ⟹ *S* ∈ *C* ⟹ *extend S C f n* ∈ *C*›
by (*induct n*) *simp-all*

theorem *Extend-in-C*: ‹*consistency C* ⟹ *finite-char C* ⟹ *S* ∈ *C* ⟹ *Extend S C f* ∈ *C*›
using *chain-union-closed is-chain-extend extend-in-C* **unfolding** *Extend-def*
by *blast*

theorem *Extend-subset*: ‹*S* ⊆ *Extend S C f*›
unfolding *Extend-def* **using** *Union-upper extend.simps*(*1*) *range-eqI*
by *metis*

definition *maximal* :: ‹*'a set* ⇒ *'a set set* ⇒ *bool*› **where**
‹*maximal S C* ≡ ∀ *S′* ∈ *C*. *S* ⊆ *S′* ⟶ *S* = *S′*›

theorem *Extend-maximal*:

assumes ‹$\forall\, y :: 'i\ fm.\ \exists\, n.\ y = f\ n$› ‹*finite-char C*›
shows ‹*maximal* (*Extend S C f*) *C*›
unfolding *maximal-def Extend-def*
proof (*intro ballI impI*)
fix S'
assume ‹$S' \in C$› ‹$(\bigcup x.\ extend\ S\ C\ f\ x) \subseteq S'$›
moreover have ‹$S' \subseteq (\bigcup x.\ extend\ S\ C\ f\ x)$›
proof (*rule ccontr*)
assume ‹$\neg\ S' \subseteq (\bigcup x.\ extend\ S\ C\ f\ x)$›
then obtain *z* **where** ‹$z \in S'$› ‹$z \notin (\bigcup x.\ extend\ S\ C\ f\ x)$›
by *blast*
then obtain *n* **where** ‹$z = f\ n$›
using ‹$\forall\, y.\ \exists\, n.\ y = f\ n$›
by *blast*

have ‹$extend\ S\ C\ f\ n \cup \{f\ n\} \subseteq S'$›
using ‹$(\bigcup x.\ extend\ S\ C\ f\ x) \subseteq S'$› ‹$z = f\ n$› ‹$z \in S'$›
by *blast*

have ‹*subset-closed C*›
using ‹*finite-char C*› *finite-char-closed*
by *blast*
then have ‹$\forall\, S' \in C.\ \forall\, S \subseteq S'.\ S \in C$›
unfolding *subset-closed-def*
by *simp*
then have ‹$\forall\, S \subseteq S'.\ S \in C$›
using ‹$S' \in C$›
by *blast*
then have ‹$extend\ S\ C\ f\ n \cup \{f\ n\} \in C$›
using ‹$extend\ S\ C\ f\ n \cup \{f\ n\} \subseteq S'$›
by *blast*
then have ‹$z \in extend\ S\ C\ f\ (Suc\ n)$›
using ‹$z \notin (\bigcup x.\ extend\ S\ C\ f\ x)$› ‹$z = f\ n$›
by *simp*
then show *False*
using ‹$z \notin (\bigcup x.\ extend\ S\ C\ f\ x)$›
by *blast*
qed
ultimately show ‹$(\bigcup x.\ extend\ S\ C\ f\ x) = S'$›
by *simp*
qed

8.4 K consistency

theorem *K-consistency*: ‹*consistency* $\{set\ G \mid G.\ \neg \vdash imply\ G\ \bot\}$›
unfolding *consistency-def*
proof (*intro conjI ballI allI impI notI*)
fix *S* :: ‹*'i fm set*›
assume ‹$S \in \{set\ G \mid G.\ \neg \vdash imply\ G\ \bot\}$› (**is** ‹$S \in ?C$›)

then obtain G **where** $*$: ‹$S = set\ G$› **and** ‹$\neg \vdash imply\ G\ \bot$›
by *blast*

{ fix i
assume ‹$Pro\ i \in S \wedge (\neg\ Pro\ i) \in S$›
then have ‹$\vdash imply\ G\ (Pro\ i)$› ‹$\vdash imply\ G\ (\neg\ Pro\ i)$›
using *K-imply-member* $*$ **by** *blast+*
then have ‹$\vdash imply\ G\ \bot$›
using *K-FFI K-right-mp* **by** *blast*
then show *False*
using ‹$\neg \vdash imply\ G\ \bot$› **by** *blast* **}**

{ assume ‹$\bot \in S$›
then have ‹$\vdash imply\ G\ \bot$›
using *K-imply-member* $*$ **by** *blast*
then show *False*
using ‹$\neg \vdash imply\ G\ \bot$› **by** *blast* **}**

{ fix Z
assume ‹$(\neg\ \neg\ Z) \in S$›
then have ‹$\vdash imply\ G\ (\neg\ \neg\ Z)$›
using *K-imply-member* $*$ **by** *simp*
then have ‹$\neg \vdash imply\ (Z\ \#\ G)\ \bot$›
using *K-ImpI K-right-mp* ‹$\neg \vdash imply\ G\ \bot$› **by** *blast*
moreover have ‹$S \cup \{Z\} = set\ (Z\ \#\ G)$›
using $*$ **by** *simp*
ultimately show ‹$S \cup \{Z\} \in ?C$›
by *blast* **}**

{ fix $A\ B$
assume ‹$(A \wedge B) \in S$›
then have ‹$\vdash imply\ G\ (A \wedge B)$›
using *K-imply-member* $*$ **by** *simp*
moreover have ‹$\vdash ((A \wedge B) \longrightarrow A)$› ‹$\vdash ((A \wedge B) \longrightarrow B)$›
using *A1* **by** *force+*
ultimately have ‹$\vdash imply\ G\ A$› ‹$\vdash imply\ G\ B$›
using *K-right-mp K-imply-head R1* **by** (*metis imply.simps*(*2*))*+*
then have ‹$\neg \vdash imply\ (A\ \#\ B\ \#\ G)\ \bot$›
using *K-imply-Cons* ‹$\neg \vdash imply\ G\ \bot$› *cut* **by** *metis*
moreover have ‹$S \cup \{A,\ B\} = set\ (A\ \#\ B\ \#\ G)$›
using $*$ **by** *simp*
ultimately show ‹$S \cup \{A,\ B\} \in ?C$›
by *blast* **}**

{ fix $A\ B$
assume ‹$(\neg\ (A \vee B)) \in S$›
then have ‹$\vdash imply\ ((\neg\ B)\ \#\ G)\ (\neg\ (A \vee B))$› ‹$\vdash imply\ G\ (\neg\ (A \vee B))$›
using $*$ *K-imply-Cons K-imply-member* **by** *blast+*
moreover have ‹$\vdash imply\ ((\neg\ B)\ \#\ G)\ (\neg\ (A \vee B) \longrightarrow \neg\ A)$› ‹$\vdash imply\ G\ (\neg$

$(A \vee B) \longrightarrow \neg B)$›
by (*simp-all add*: *A1 tautology-imply*)
ultimately have ‹$\vdash$ *imply* ((¬ *B*) # *G*) (¬ *A*)› ‹$\vdash$ *imply* *G* (¬ *B*)›
using *K-right-mp* **by** *blast+*

then have ‹¬ $\vdash$ *imply* ((¬ *A*) # (¬ *B*) # *G*) ⊥›
using ‹¬ $\vdash$ *imply* *G* ⊥› *cut* **by** *blast*
moreover have ‹*S* ∪ {¬ *A*, ¬ *B*} = *set* ((¬ *A*) # (¬ *B*) # *G*)›
using ∗ **by** *simp*
ultimately show ‹*S* ∪ {¬ *A*, ¬ *B*} ∈ *?C*›
by *blast* **}**

{ fix *A B*
assume ‹(¬ (*A* ⟶ *B*)) ∈ *S*›
then have ‹$\vdash$ *imply* ((¬ *B*) # *G*) (¬ (*A* ⟶ *B*))› ‹$\vdash$ *imply* *G* (¬ (*A* ⟶ *B*))›
using ∗ *K-imply-member K-imply-Cons* **by** *blast+*
moreover have
‹$\vdash$ *imply* ((¬ *B*) # *G*) (¬ (*A* ⟶ *B*) ⟶ *A*)›
‹$\vdash$ *imply* *G* (¬ (*A* ⟶ *B*) ⟶ ¬ *B*)›
by (*simp-all add*: *A1 tautology-imply*)
ultimately have ‹$\vdash$ *imply* ((¬ *B*) # *G*) *A*› ‹$\vdash$ *imply* *G* (¬ *B*)›
using *K-right-mp* **by** *blast+*

then have ‹¬ $\vdash$ *imply* (*A* # (¬ *B*) # *G*) ⊥›
using ‹¬ $\vdash$ *imply* *G* ⊥› *cut* **by** *blast*
moreover have ‹*S* ∪ {*A*, ¬ *B*} = *set* (*A* # (¬ *B*) # *G*)›
using ∗ **by** *simp*
ultimately show ‹*S* ∪ {*A*, ¬ *B*} ∈ *?C*›
by *blast* **}**

{ fix *A B*
assume ‹(*A* ∨ *B*) ∈ *S*›
then have ‹$\vdash$ *imply* *G* (*A* ∨ *B*)›
using ∗ *K-imply-member* **by** *simp*

{ assume ‹(∀ *G'*. *set* *G'* = *S* ∪ {*A*} ⟶ $\vdash$ *imply* *G'* ⊥)›
and ‹(∀ *G'*. *set* *G'* = *S* ∪ {*B*} ⟶ $\vdash$ *imply* *G'* ⊥)›
then have ‹$\vdash$ *imply* (*A* # *G*) ⊥› ‹$\vdash$ *imply* (*B* # *G*) ⊥›
using ∗ **by** (*metis Un-insert-right list.simps*(*15*) *sup-bot.right-neutral*)+
then have ‹$\vdash$ *imply* *G* ⊥›
using ‹$\vdash$ *imply* *G* (*A* ∨ *B*)› *K-DisE* **by** *blast*
then have *False*
using ‹¬ $\vdash$ *imply* *G* ⊥› **by** *blast* **}**
then show ‹*S* ∪ {*A*} ∈ *?C* ∨ *S* ∪ {*B*} ∈ *?C*›
by *blast* **}**

{ fix *A B*
assume ‹(¬ (*A* ∧ *B*)) ∈ *S*›
then have ‹$\vdash$ *imply* *G* (¬ (*A* ∧ *B*))›

using ∗ *K-imply-member* **by** *blast*
moreover have ‹⊢ *imply G* (¬ (*A* ∧ *B*) ⟶ (¬ *A*) ∨ (¬ *B*))›
by (*simp add*: *A1 tautology-imply*)
ultimately have ‹⊢ *imply G* ((¬ *A*) ∨ (¬ *B*))›
using *K-right-mp* **by** *blast*

{ assume ‹(∀ *G'*. *set G'* = *S* ∪ {¬ *A*} ⟶ ⊢ *imply G'* ⊥)›
and ‹(∀ *G'*. *set G'* = *S* ∪ {¬ *B*} ⟶ ⊢ *imply G'* ⊥)›
then have ‹⊢ *imply* ((¬ *A*) # *G*) ⊥› ‹⊢ *imply* ((¬ *B*) # *G*) ⊥›
using ∗ **by** (*metis Un-insert-right list.simps*(*15*) *sup-bot.right-neutral*)+
then have ‹⊢ *imply G* ⊥›
using *K-DisE* ‹⊢ *imply G* ((¬ *A*) ∨ (¬ *B*))› **by** *blast*
then have *False*
using ‹¬ ⊢ *imply G* ⊥› **by** *blast* **}**
then show ‹*S* ∪ {¬ *A*} ∈ *?C* ∨ *S* ∪ {¬ *B*} ∈ *?C*›
by *blast* **}**

{ fix *A B*
assume ‹(*A* ⟶ *B*) ∈ *S*›
then have ‹⊢ *imply G* (*A* ⟶ *B*)›
using ∗ *K-imply-member* **by** *blast*
moreover have ‹⊢ *imply G* ((*A* ⟶ *B*) ⟶ (¬ *A*) ∨ *B*)›
by (*simp add*: *A1 tautology-imply*)
ultimately have ‹⊢ *imply G* ((¬ *A*) ∨ *B*)›
using *K-right-mp* **by** *blast*

{ assume ‹(∀ *G'*. *set G'* = *S* ∪ {¬ *A*} ⟶ ⊢ *imply G'* ⊥)›
and ‹(∀ *G'*. *set G'* = *S* ∪ {*B*} ⟶ ⊢ *imply G'* ⊥)›
then have ‹⊢ *imply* ((¬ *A*) # *G*) ⊥› ‹⊢ *imply* (*B* # *G*) ⊥›
using ∗ **by** (*metis Un-insert-right list.simps*(*15*) *sup-bot.right-neutral*)+
then have ‹⊢ *imply G* ⊥›
using *K-DisE* ‹⊢ *imply G* ((¬ *A*) ∨ *B*)› **by** *blast*
then have *False*
using ‹¬ ⊢ *imply G* ⊥› **by** *blast* **}**
then show ‹*S* ∪ {¬ *A*} ∈ *?C* ∨ *S* ∪ {*B*} ∈ *?C*›
by *blast* **}**

{ fix *A* :: ‹*'i fm*›
assume ‹*tautology A*›
then have ‹⊢ *imply G A*›
using *tautology-imply A1* **by** *blast*
then have ‹⊢ *imply* (*A* # *G*) ⊥ ⟹ ⊢ *imply G* ⊥›
using *cut* **by** *blast*
moreover have ‹*S* ∪ {*A*} = *set* (*A* # *G*)›
using ∗ **by** *simp*
ultimately show ‹*S* ∪ {*A*} ∈ *?C*›
using ‹¬ ⊢ *imply G* ⊥› **by** *blast* **}**

{ fix *A i*

assume ‹$K\ i\ A \in S \wedge (\neg\ K\ i\ A) \in S$›
then have ‹$\vdash$ *imply* G $(K\ i\ A)$› ‹$\vdash$ *imply* G $(\neg\ K\ i\ A)$›
using *K-imply-member* $*$ **by** *blast+*
then have ‹$\vdash$ *imply* $G\ \bot$›
using *K-FFI K-right-mp* **by** *blast*
then show *False*
using ‹$\neg \vdash$ *imply* $G\ \bot$› **by** *blast* **}**

qed

theorem *K-finite-consistency*: ‹*consistency* (*mk-finite-char* (*close* {*set* G | G. $\neg \vdash$ *imply* $G\ \bot$}))›
using *K-consistency finite-consistency close-closed close-consistency* **by** *blast*

theorem *K-concrete-finite-consistency*:
defines ‹$C \equiv$ *mk-finite-char* (*close* {*set* G | G. $\neg \vdash$ *imply* $G\ \bot$})›
assumes ‹*set* $G \in C$›
shows ‹$\neg \vdash$ *imply* $G\ \bot$›
using *assms* **unfolding** *C-def mk-finite-char-def close-def*
using *K-imply-weaken* **by** *fastforce*

9 Model existence

lemma *at-least-one-in-maximal*:
assumes ‹*consistency* C› ‹$V \in C$› ‹*maximal* V C›
shows ‹$p \in V \vee (\neg\ p) \in V$›
proof −
have ‹$V \cup \{p\} \in C \vee V \cup \{\neg\ p\} \in C$›
proof (*rule ccontr*)
assume ‹$\neg\ (V \cup \{p\} \in C \vee V \cup \{\neg\ p\} \in C)$›
then have ‹$V \cup \{p \vee \neg\ p\} \notin C$›
using *assms* **unfolding** *consistency-def maximal-def*
by (*metis* (*no-types*, *hide-lams*) *insert-iff subset-iff sup.absorb-iff1 sup.cobounded1*)
then show *False*
using *assms* **unfolding** *consistency-def maximal-def*
by *simp*
qed
then show *?thesis*
using ‹*maximal* V C› **unfolding** *maximal-def*
by *fast*
qed

lemma *at-most-one-in-maximal*:
assumes ‹*consistency* C› ‹$V \in C$› ‹*maximal* V C›
shows ‹$\neg\ (p \in V \wedge (\neg\ p) \in V)$›
proof (*induct* p)
case *FF*
then show *?case*
using *assms* **unfolding** *consistency-def* **by** *blast*

next
case (*Pro x*)
then show *?case*
using *assms* **unfolding** *consistency-def* **by** *simp*
next
case (*Dis p q*)
show *?case*
proof
assume ‹$(p \vee q) \in V \wedge (\neg (p \vee q)) \in V$›
then have ‹$V \cup \{p\} \in C \vee V \cup \{q\} \in C$› ‹$V \cup \{\neg p, \neg q\} \in C$›
using ‹*consistency C*› ‹$V \in C$› **unfolding** *consistency-def* **by** *simp-all*
then have ‹$p \in V \vee q \in V$› ‹$(\neg p) \in V$› ‹$(\neg q) \in V$›
using ‹*maximal V C*› **unfolding** *maximal-def* **by** *fast+*
then show *False*
using *Dis* **by** *blast*
qed
next
case (*Con p q*)
show *?case*
proof
assume ‹$(p \wedge q) \in V \wedge (\neg (p \wedge q)) \in V$›
then have ‹$V \cup \{p, q\} \in C$› ‹$V \cup \{\neg p\} \in C \vee V \cup \{\neg q\} \in C$›
using ‹*consistency C*› ‹$V \in C$› **unfolding** *consistency-def* **by** *simp-all*
then have ‹$p \in V$› ‹$q \in V$› ‹$(\neg p) \in V \vee (\neg q) \in V$›
using ‹*maximal V C*› **unfolding** *maximal-def* **by** *fast+*
then show *False*
using *Con* **by** *blast*
qed
next
case (*Imp p q*)
show *?case*
proof
assume ‹$(p \longrightarrow q) \in V \wedge (\neg (p \longrightarrow q)) \in V$›
then have ‹$V \cup \{\neg p\} \in C \vee V \cup \{q\} \in C$› ‹$V \cup \{p, \neg q\} \in C$›
using ‹*consistency C*› ‹$V \in C$› **unfolding** *consistency-def* **by** *simp-all*
then have ‹$(\neg p) \in V \vee q \in V$› ‹$p \in V$› ‹$(\neg q) \in V$›
using ‹*maximal V C*› **unfolding** *maximal-def* **by** *fast+*
then show *False*
using *Imp* **by** *blast*
qed
next
case (*K i p*)
then show *?case*
using ‹*consistency C*› ‹$V \in C$› **unfolding** *consistency-def*
by *simp*
qed

theorem *exactly-one-in-maximal*:
assumes ‹*consistency C*› ‹$V \in C$› ‹*maximal V C*›

shows ‹$(p \in V) \neq ((\neg\ p) \in V)$›
using *assms at-least-one-in-maximal at-most-one-in-maximal* **by** *blast*

theorem *conjuncts-in-maximal*:
assumes ‹*consistency C*› ‹$V \in C$› ‹*maximal V C*›
shows ‹$(p \wedge q) \in V \longleftrightarrow p \in V \wedge q \in V$›
proof
assume ‹$(p \wedge q) \in V$›
have ‹$V \cup \{p,\ q\} \in C$›
using ‹$(p \wedge q) \in V$› ‹*consistency C*› ‹$V \in C$› **unfolding** *consistency-def*
by *simp*
then show ‹$p \in V \wedge q \in V$›
using ‹*maximal V C*› **unfolding** *maximal-def*
by *fast*
next
assume ‹$p \in V \wedge q \in V$›
show ‹$(p \wedge q) \in V$›
proof (*rule ccontr*)
assume ‹$(p \wedge q) \notin V$›
then have ‹$(\neg\ (p \wedge q)) \in V$›
using *at-least-one-in-maximal assms* **by** *blast*
then have ‹$V \cup \{\neg\ p\} \in C \vee V \cup \{\neg\ q\} \in C$›
using ‹*consistency C*› ‹$V \in C$› **unfolding** *consistency-def*
by *simp*
then have ‹$(\neg\ p) \in V \vee (\neg\ q) \in V$›
using ‹*maximal V C*› **unfolding** *maximal-def*
by *fast*
then show *False*
using ‹$p \in V \wedge q \in V$› *assms at-most-one-in-maximal*
by *metis*
qed
qed

theorem *consequent-in-maximal*:
assumes ‹*consistency C*› ‹$V \in C$› ‹*maximal V C*› ‹$p \in V$› ‹$(p \longrightarrow q) \in V$›
shows ‹$q \in V$›
proof −
consider (*np*) ‹$V \cup \{\neg\ p\} \in C$› | (*q*) ‹$V \cup \{q\} \in C$›
using ‹$(p \longrightarrow q) \in V$› ‹*consistency C*› ‹$V \in C$› **unfolding** *consistency-def*
by *metis*
then show *?thesis*
proof *cases*
case *np*
then have ‹$(\neg\ p) \in V$›
using ‹*maximal V C*› **unfolding** *maximal-def*
by *fast*
then show *?thesis*
using *assms at-most-one-in-maximal*
by *blast*

```
  next
    case q
    then show ?thesis
      using ⟨maximal V C⟩ unfolding maximal-def
      by fast
  qed
qed

lemma K-not-neg-in-consistency: ⟨¬ ⊢ (¬ p) ⟹ {p} ∈ {set G | G. ¬ ⊢ imply G
⊥}⟩
proof −
  assume ⟨¬ ⊢ (¬ p)⟩
  moreover have ⟨set [p] = {p}⟩
    by simp
  moreover have ⟨imply [p] ⊥ = (¬ p)⟩
    by simp
  ultimately show ?thesis
    by fastforce
qed

lemma K-inconsistent-neg:
  defines ⟨C ≡ mk-finite-char (close {set G | G. ¬ ⊢ imply G ⊥})⟩
  assumes ⟨{p} ∉ C⟩
  shows ⟨⊢ (¬ p)⟩
  using assms unfolding C-def mk-finite-char-def close-def
  using K-not-neg-in-consistency by fastforce

lemma conjuncts-in-consistency:
  assumes ⟨consistency C⟩ ⟨subset-closed C⟩ ⟨S ∪ {p ∧ q} ∈ C⟩
  shows ⟨S ∪ {p, q} ∈ C⟩
proof −
  let ?S = ⟨S ∪ {p ∧ q}⟩
  have ⟨?S ∈ C⟩
    using assms by blast
  moreover have ⟨(p ∧ q) ∈ ?S⟩
    by simp
  ultimately have ⟨?S ∪ {p, q} ∈ C⟩
    using assms unfolding consistency-def by simp
  moreover have ⟨S ∪ {p, q} ⊆ ?S ∪ {p, q}⟩
    by blast
  ultimately show ⟨S ∪ {p, q} ∈ C⟩
    using ⟨subset-closed C⟩ unfolding subset-closed-def by simp
qed

lemma conjoined-in-consistency:
  assumes ⟨consistency C⟩ ⟨subset-closed C⟩ ⟨S ∪ {conjoin ps q} ∈ C⟩
  shows ⟨S ∪ set ps ∪ {q} ∈ C⟩
  using ⟨S ∪ {conjoin ps q} ∈ C⟩
proof (induct ps arbitrary: S)
```

```
  case Nil
  then show ?case
    using ‹subset-closed C› unfolding subset-closed-def by simp
next
  case (Cons p ps)
  then have ‹S ∪ {p, conjoin ps q} ∈ C›
    using assms conjuncts-in-consistency by auto
  then have ‹S ∪ {p} ∪ {conjoin ps q} ∈ C›
    by (metis insert-is-Un sup-assoc)
  then show ?case
    using Cons by fastforce
qed

lemma inconsistent-conjoin:
  defines ‹C ≡ mk-finite-char (close {set G | G. ¬ ⊢ imply G ⊥})›
  assumes ‹set G ∪ {p} ∉ C›
  shows ‹⊢ (¬ conjoin G p)›
proof (rule ccontr)
  have ‹consistency C›
    unfolding C-def using K-finite-consistency by auto
  have ‹subset-closed C›
    unfolding C-def by (simp add: finite-char finite-char-closed)

  assume ‹¬ ⊢ (¬ conjoin G p)›
  then have ‹{conjoin G p} ∈ C›
    unfolding C-def using K-inconsistent-neg by blast
  then have ‹set G ∪ {p} ∈ C›
    using conjoined-in-consistency ‹consistency C› ‹subset-closed C› by fastforce
  then show False
    using assms(2) by blast
qed

theorem K-in-maximal:
  defines ‹C ≡ mk-finite-char (close {set G | G. ¬ ⊢ imply G ⊥})›
  assumes ‹⊢ p› ‹V ∈ C› ‹maximal V C›
  shows ‹p ∈ V›
proof −
  have ‹consistency C›
    unfolding C-def using K-finite-consistency by auto
  have ‹subset-closed C›
    unfolding C-def by (simp add: finite-char finite-char-closed)

  have ‹⊢ (p ⟶ ¬ ¬ p)›
    by (simp add: A1)
  then have ‹⊢ (¬ ¬ p)›
    using ‹⊢ p› R1 by blast

  show ?thesis
  proof (rule ccontr)
```

assume ‹$p \notin V$›
then have ‹$(\neg\ p) \in V$›
using *at-least-one-in-maximal* ‹*consistency C*› ‹$V \in C$› ‹*maximal V C*› **by** *blast*
then have ‹*set* $[\neg\ p] \in C$›
using ‹$V \in C$› ‹*subset-closed C*› **unfolding** *subset-closed-def* **by** *simp*
then have ‹$\neg \vdash (\neg\ \neg\ p)$›
using *C-def K-concrete-finite-consistency* **by** *fastforce*
then show *False*
using ‹$\vdash (\neg\ \neg\ p)$› **by** *blast*
qed
qed

lemma *exists-finite-inconsistent*:
fixes *C* :: ‹*'i fm set set*›
assumes ‹*finite-char C*› ‹$V \cup \{\neg\ p\} \notin C$›
shows ‹$\exists\ W.\ W \cup \{\neg\ p\} \subseteq V \cup \{\neg\ p\} \wedge (\neg\ p) \notin W \wedge$ *finite W* $\wedge\ W \cup \{\neg\ p\} \notin C$›
using *assms* **unfolding** *finite-char-def*
by (*smt Un-insert-right Un-subset-iff finite-insert inf-sup-ord*(*4*) *mk-disjoint-insert sup-bot.comm-neutral sup-ge1*)

theorem *exists-maximal-superset*:
fixes *C* :: ‹(*'i* :: *countable*) *fm set set*›
assumes ‹*consistency C*› ‹*finite-char C*› ‹$V \in C$›
obtains *W* **where** ‹$V \subseteq W$› ‹ $W \in C$› ‹*maximal W C*›
proof −
let *?W* = ‹*Extend V C from-nat*›

have ‹$V \subseteq$ *?W*›
using *Extend-subset* **by** *blast*
moreover have ‹*?W* $\in C$›
using *assms Extend-in-C* **by** *blast*
moreover have ‹*maximal ?W C*›
using *assms Extend-maximal*
by (*metis from-nat-to-nat*)
ultimately show *?thesis*
using *that* **by** *blast*
qed

type-synonym *'i s-max* = ‹*'i fm set*›

abbreviation *pi* :: ‹*'i s-max* ⇒ *id* ⇒ *bool*› **where**
‹*pi s i* ≡ *Pro i* ∈ *s*›

abbreviation *partition* :: ‹*'i fm set* ⇒ *'i* ⇒ *'i fm set*› **where**
‹*partition V i* ≡ {*p*. *K i p* ∈ *V*}›

abbreviation *reach* :: ‹*'i fm set set* ⇒ *'i* ⇒ *'i s-max* ⇒ *'i s-max set*› **where**

```
  ‹reach C i V ≡ {W. partition V i ⊆ W ∧ W ∈ C ∧ maximal W C}›

theorem model-existence:
  fixes p :: ‹('i :: countable) fm›
  defines ‹C ≡ mk-finite-char (close {set G | G. ¬ ⊢ imply G ⊥})›
  assumes ‹V ∈ C› ‹maximal V C›
  shows ‹(p ∈ V ⟷ Kripke pi (reach C), V ⊨ p) ∧
    ((¬ p) ∈ V ⟷ Kripke pi (reach C), V ⊨ ¬ p)›
proof −
  have ‹consistency C›
    unfolding C-def using K-finite-consistency by auto
  have ‹finite-char C›
    unfolding C-def by (simp add: finite-char)
  have ‹subset-closed C›
    unfolding C-def by (simp add: finite-char finite-char-closed)

  show ?thesis
    using ‹V ∈ C› ‹maximal V C›
  proof (induct p arbitrary: V)
    case FF
    then show ?case
    proof (intro conjI impI iffI)
      assume ‹⊥ ∈ V›
      then have False
        using ‹consistency C› ‹V ∈ C› unfolding consistency-def by blast
      then show ‹Kripke pi (reach C), V ⊨ ⊥› ..
    next
      assume ‹(¬ ⊥) ∈ V›
      show ‹Kripke pi (reach C), V ⊨ ¬ ⊥›
        by simp
    next
      assume ‹Kripke pi (reach C), V ⊨ ⊥›
      then show ‹⊥ ∈ V›
        by simp
    next
      assume ‹Kripke pi (reach C), V ⊨ ¬ ⊥›
      have ‹⊥ ∉ V›
        using ‹consistency C› ‹V ∈ C› unfolding consistency-def
        by blast
      then show ‹(¬ ⊥) ∈ V›
        using at-least-one-in-maximal FF ‹consistency C›
        by blast
    qed
  next
    case (Pro i)
    then show ?case
    proof (intro conjI impI iffI)
      assume ‹Pro i ∈ V›
      then have ‹pi V i›
```

```
    using ⟨maximal V C⟩ by blast
  then show ⟨Kripke pi (reach C), V ⊨ Pro i⟩
    by simp
next
  assume ⟨(¬ Pro i) ∈ V⟩
  then have ⟨Pro i ∉ V⟩
    using ⟨consistency C⟩ ⟨V ∈ C⟩ unfolding consistency-def by fast
  then have ⟨¬ (pi V i)⟩
    using ⟨maximal V C⟩ by blast
  then show ⟨Kripke pi (reach C), V ⊨ ¬ Pro i⟩
    by simp
next
  assume ⟨Kripke pi (reach C), V ⊨ Pro i⟩
  then show ⟨Pro i ∈ V⟩
    by simp
next
  assume ⟨Kripke pi (reach C), V ⊨ ¬ Pro i⟩
  then have ⟨Pro i ∉ V⟩
    by simp
  then show ⟨(¬ Pro i) ∈ V⟩
    using at-least-one-in-maximal Pro ⟨consistency C⟩
    by blast
qed
next
  case (Dis p q)
  have ⟨(p ∨ q) ∈ V ⟶ Kripke pi (reach C), V ⊨ (p ∨ q)⟩
  proof
    assume ⟨(p ∨ q) ∈ V⟩
    then have ⟨V ∪ {p} ∈ C ∨ V ∪ {q} ∈ C⟩
      using ⟨consistency C⟩ ⟨V ∈ C⟩ unfolding consistency-def by simp
    then consider ⟨p ∈ V⟩ | ⟨q ∈ V⟩
      using ⟨maximal V C⟩ unfolding maximal-def by fast
    then show ⟨Kripke pi (reach C), V ⊨ (p ∨ q)⟩
      using Dis by cases simp-all
  qed
  moreover have ⟨(¬ (p ∨ q)) ∈ V ⟶ Kripke pi (reach C), V ⊨ ¬ (p ∨ q)⟩
  proof
    assume ⟨(¬ (p ∨ q)) ∈ V⟩
    then have ⟨V ∪ {¬ p, ¬ q} ∈ C⟩
      using ⟨consistency C⟩ ⟨V ∈ C⟩ unfolding consistency-def by simp
    then have ⟨(¬ p) ∈ V ∧ (¬ q) ∈ V⟩
      using ⟨maximal V C⟩ unfolding maximal-def by fast
    then show ⟨Kripke pi (reach C), V ⊨ ¬ (p ∨ q)⟩
      using Dis by simp
  qed
  ultimately show ?case
    using at-least-one-in-maximal Dis ⟨consistency C⟩
    by auto
next
```

```
    case (Con p q)
    have ‹(p ∧ q) ∈ V ⟶ Kripke pi (reach C), V ⊨ (p ∧ q)›
    proof
      assume ‹(p ∧ q) ∈ V›
      then have ‹V ∪ {p, q} ∈ C›
        using ‹consistency C› ‹V ∈ C› unfolding consistency-def by simp
      then have ‹p ∈ V› ‹q ∈ V›
        using ‹maximal V C› unfolding maximal-def by fast+
      then show ‹Kripke pi (reach C), V ⊨ (p ∧ q)›
        using Con by simp
    qed
    moreover have ‹(¬ (p ∧ q)) ∈ V ⟶ Kripke pi (reach C), V ⊨ ¬ (p ∧ q)›
    proof
      assume ‹(¬ (p ∧ q)) ∈ V›
      then have ‹V ∪ {¬ p} ∈ C ∨ V ∪ {¬ q} ∈ C›
        using ‹consistency C› ‹V ∈ C› unfolding consistency-def by simp
      then consider ‹(¬ p) ∈ V› | ‹(¬ q) ∈ V›
        using ‹maximal V C› unfolding maximal-def by fast
      then show ‹Kripke pi (reach C), V ⊨ ¬ (p ∧ q)›
        using Con by cases simp-all
    qed
    ultimately show ?case
      using at-least-one-in-maximal Con ‹consistency C›
      by auto
  next
    case (Imp p q)
    have ‹(p ⟶ q) ∈ V ⟶ Kripke pi (reach C), V ⊨ (p ⟶ q)›
    proof
      assume ‹(p ⟶ q) ∈ V›
      then have ‹V ∪ {¬ p} ∈ C ∨ V ∪ {q} ∈ C›
        using ‹consistency C› ‹V ∈ C› unfolding consistency-def by simp
      then consider ‹(¬ p) ∈ V› | ‹q ∈ V›
        using ‹maximal V C› unfolding maximal-def by fast
      then show ‹Kripke pi (reach C), V ⊨ (p ⟶ q)›
        using Imp by cases simp-all
    qed
    moreover have ‹(¬ (p ⟶ q)) ∈ V ⟶ Kripke pi (reach C), V ⊨ ¬ (p ⟶
q)›
    proof
      assume ‹(¬ (p ⟶ q)) ∈ V›
      then have ‹V ∪ {p, ¬ q} ∈ C›
        using ‹consistency C› ‹V ∈ C› unfolding consistency-def by simp
      then have ‹p ∈ V ∧ (¬ q) ∈ V›
        using ‹maximal V C› unfolding maximal-def by fast
      then show ‹Kripke pi (reach C), V ⊨ ¬ (p ⟶ q)›
        using Imp by simp
    qed
    ultimately show ?case
      using at-least-one-in-maximal Imp ‹consistency C›
```

```
    by auto
  next
    case (K i p)
    have ‹K i p ∈ V ⟶ Kripke pi (reach C), V ⊨ K i p›
    proof
      assume ‹K i p ∈ V›
      then have ‹∀ W ∈ reach C i V. p ∈ W ∧ W ∈ C ∧ maximal W C›
        using ‹consistency C› by blast
      then have ‹∀ W ∈ reach C i V. Kripke pi (reach C), W ⊨ p›
        using K by simp
      then show ‹Kripke pi (reach C), V ⊨ K i p›
        by simp
    qed
    moreover have ‹(Kripke pi (reach C), V ⊨ K i p) ⟶ K i p ∈ V›
    proof (intro allI impI)
      assume ‹Kripke pi (reach C), V ⊨ K i p›

      have ‹partition V i ∪ {¬ p} ∉ C›
      proof
        assume ‹partition V i ∪ {¬ p} ∈ C›
        then obtain W where ‹partition V i ∪ {¬ p} ⊆ W› ‹W ∈ C› ‹maximal
W C›
          using ‹consistency C› ‹finite-char C› exists-maximal-superset by blast
        then have ‹Kripke pi (reach C), W ⊨ ¬ p›
          using K ‹consistency C› ‹finite-char C› by blast
        moreover have ‹W ∈ reach C i V›
          using ‹partition V i ∪ {¬ p} ⊆ W› ‹W ∈ C› ‹maximal W C›
          by simp
        ultimately have ‹Kripke pi (reach C), V ⊨ ¬ K i p›
          by auto
        then show False
          using ‹Kripke pi (reach C), V ⊨ K i p›
          by auto
      qed

      then obtain W where
        ‹W ∪ {¬ p} ⊆ partition V i ∪ {¬ p}› ‹(¬ p) ∉ W› ‹finite W› ‹W ∪ {¬ p}
∉ C›
        using ‹finite-char C› exists-finite-inconsistent by force

      obtain L where ‹set L = W›
        using ‹finite W› finite-list by blast

      then have ‹⊢ (¬ conjoin L (¬ p))›
        using inconsistent-conjoin ‹W ∪ {¬ p} ∉ C› C-def by blast
      then have ‹⊢ imply L p›
        using K-conjoin-imply by blast
      then have ‹⊢ K i (imply L p)›
        using R2 by fast
```

```
      then have ‹⊢ imply (map (K i) L) (K i p)›
        using K-distrib-K-imp by fast
      then have ‹imply (map (K i) L) (K i p) ∈ V›
        using K-in-maximal K.prems(1, 2) unfolding C-def by blast
      then show ‹K i p ∈ V›
        using ‹set L = W› ‹W ∪ {¬ p} ⊆ partition V i ∪ {¬ p}› ‹(¬ p) ∉ W›
      proof (induct L arbitrary: W)
        case Nil
        then show ?case
          by simp
      next
        case (Cons a L)
        then have ‹K i a ∈ V›
          by auto
        then have ‹imply (map (K i) L) (K i p) ∈ V›
       using Cons(2) ‹consistency C› ‹V ∈ C› ‹maximal V C› consequent-in-maximal
by auto
        then show ?case
          using Cons by auto
      qed
    qed
    moreover have ‹(Kripke pi (reach C), V ⊨ ¬ K i p) ⟶ (¬ K i p) ∈ V›
       using K.prems(1) ‹consistency C› ‹maximal V C› at-least-one-in-maximal
calculation(1) by auto
    moreover have ‹(¬ K i p) ∈ V ⟶ Kripke pi (reach C), V ⊨ ¬ K i p›
    using K.prems(1) ‹consistency C› ‹maximal V C› calculation(2) exactly-one-in-maximal
by auto
    ultimately show ?case
      by blast
  qed
qed
```

9.1 Completeness

```
lemma imply-completeness:
  assumes valid: ‹∀(M :: ('i, ('i :: countable) fm set) kripke) s.
    list-all (λq. M, s ⊨ q) G ⟶ M, s ⊨ p›
  shows ‹⊢ imply G p›
proof (rule K-Boole, rule ccontr)
  assume ‹¬ ⊢ imply ((¬ p) # G) ⊥›

  let ?S = ‹set ((¬ p) # G)›
  let ?C = ‹mk-finite-char (close {set G | G. ¬ ⊢ imply G ⊥})›
  let ?V = ‹Extend ?S ?C from-nat›
  let ?M = ‹Kripke pi (reach ?C)›

  have ‹consistency ?C›
    by (simp add: K-finite-consistency)
  have ‹subset-closed ?C›
```

using *finite-char finite-char-closed* **by** *blast*
have ‹*finite-char ?C*›
using *finite-char* **by** *blast*

have ‹*?S* ∈ *?C*›
using ‹¬ ⊢ *imply* ((¬ *p*) # *G*) ⊥› *close-closed close-subset finite-char-subset*
by *fast*
then have ‹*?V* ∈ *?C*›
using *Extend-in-C K-finite-consistency* ‹*finite-char ?C*› **by** *blast*

have ‹∀ *y* :: *'i fm*. ∃ *n*. *y* = *from-nat n*›
by (*metis from-nat-to-nat*)
then have ‹*maximal ?V ?C*›
using *Extend-maximal* ‹*finite-char ?C*› **by** *blast*

{ fix *x*
assume ‹*x* ∈ *?S*›
then have ‹*x* ∈ *?V*›
using *Extend-subset* **by** *blast*
then have ‹*?M*, *?V* ⊨ *x*›
using *model-existence* ‹*?V* ∈ *?C*› ‹*maximal ?V ?C*› **by** *blast* **}**
then have ‹*?M*, *?V* ⊨ (¬ *p*)› ‹*list-all* (λ*p*. *?M*, *?V* ⊨ *p*) *G*›
unfolding *list-all-def* **by** *fastforce+*
then have ‹*?M*, *?V* ⊨ *p*›
using *valid* **by** *blast*
then show *False*
using ‹*?M*, *?V* ⊨ (¬ *p*)› **by** *simp*
qed

theorem *completeness*:
assumes ‹∀ (*M* :: (*'i* :: *countable*, *'i fm set*) *kripke*) *s*. *M*, *s* ⊨ *p*›
shows ‹⊢ *p*›
using *assms imply-completeness*[**where** *G*=‹[]›] **by** *auto*

10 Main Result

— System K is sound and complete

abbreviation ‹*valid p* ≡ ∀ (*M* :: (*nat*, *nat s-max*) *kripke*) *s*. *M*, *s* ⊨ *p*›

theorem *main*: ‹*valid p* ⟷ ⊢ *p*›
proof
assume ‹*valid p*›
with *completeness* **show** ‹⊢ *p*›
by *blast*
next
assume ‹⊢ *p*›
with *soundness* **show** ‹*valid p*›
by (*intro allI*)

qed

corollary ‹*valid p* ⟶ *M*, *s* ⊨ *p*›
proof
  assume ‹*valid p*›
  then have ‹⊢ *p*›
    unfolding *main* **.**
  with *soundness* **show** ‹*M*, *s* ⊨ *p*› **.**
qed

11 Acknowledgements

The definition of a consistency property, subset closure, finite character and maximally consistent sets is based on work by Berghofer, but has been adapted from first-order logic to epistemic logic.

- Stefan Berghofer: First-Order Logic According to Fitting. https://www.isa-afp.org/entries/FOL-Fitting.shtml

end

References

[1] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.