Epistemic Logic: Completeness of Modal Logics

Asta Halkjær From

March 19, 2025

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) [2]. The extensions of system K (T, KB, K4, S4, S5) and their completeness proofs are based on the textbook "Modal Logic" by Blackburn, de Rijke and Venema (Cambridge University Press 2001) [1].

Contents

```
theory Maximal-Consistent-Sets imports HOL−Cardinals.Cardinal-Order-Relation
begin

context wo-rel begin

lemma underS-bound: ‹a ∈ underS n ⟹ b ∈ underS n ⟹ a ∈ under b ∨ b ∈
under a›
  by (meson BNF-Least-Fixpoint.underS-Field REFL Refl-under-in in-mono un-
der-ofilter ofilter-linord)

lemma finite-underS-bound:
  assumes ‹finite X› ‹X ⊆ underS n› ‹X ≠ {}›
  shows ‹∃ a ∈ X. ∀ b ∈ X. b ∈ under a›
  using assms
proof (induct X rule: finite-induct)
  case (insert x F)
  then show ?case
  proof (cases ‹F = {}›)
    case True
    then show ?thesis
      using insert underS-bound by fast
  next
    case False
    then show ?thesis
   using insert underS-bound by (metis TRANS insert-absorb insert-iff insert-subset
under-trans)
  qed
qed simp

lemma finite-bound-under:
  assumes ‹finite p› ‹p ⊆ (⋃ n ∈ Field r. f n)›
  shows ‹∃ m. p ⊆ (⋃ n ∈ under m. f n)›
    using assms
proof (induct rule: finite-induct)
  case (insert x p)
  then obtain m where ‹p ⊆ (⋃ n ∈ under m. f n)›
    by fast
  moreover obtain m′ where ‹x ∈ f m′› ‹m′ ∈ Field r›
    using insert(4) by blast
  then have ‹x ∈ (⋃ n ∈ under m′. f n)›
    using REFL Refl-under-in by fast
  ultimately have ‹{x} ∪ p ⊆ (⋃ n ∈ under m. f n) ∪ (⋃ n ∈ under m′. f n)›
    by fast
  then show ?case
     by (metis SUP-union Un-commute insert-is-Un sup.absorb-iff2 ofilter-linord
under-ofilter)
qed simp
```

end

locale *MCS-Lim-Ord* =
  fixes *r* :: ‹*'a rel*›
  assumes *WELL*: ‹*Well-order r*›
    and *isLimOrd-r*: ‹*isLimOrd r*›
  fixes *consistent* :: ‹*'a set* ⇒ *bool*›
  assumes *consistent-hereditary*: ‹*consistent S* ⟹ *S'* ⊆ *S* ⟹ *consistent S'*›
    and *inconsistent-finite*: ‹⋀*S*. ¬ *consistent S* ⟹ ∃ *S'* ⊆ *S*. *finite S'* ∧ ¬ *consistent S'*›
begin

definition *extendS* :: ‹*'a set* ⇒ *'a* ⇒ *'a set* ⇒ *'a set*› **where**
  ‹*extendS S n prev* ≡ *if consistent* ({*n*} ∪ *prev*) *then* {*n*} ∪ *prev else prev*›

definition *extendL* :: ‹(*'a* ⇒ *'a set*) ⇒ *'a* ⇒ *'a set*› **where**
  ‹*extendL rec n* ≡ ⋃ *m* ∈ *underS r n*. *rec m*›

definition *extend* :: ‹*'a set* ⇒ *'a* ⇒ *'a set*› **where**
  ‹*extend S n* ≡ *worecZSL r S* (*extendS S*) *extendL n*›

lemma *wo-rel-r*: ‹*wo-rel r*›
  by (*simp add*: *WELL wo-rel.intro*)

lemma *adm-woL-extendL*: ‹*adm-woL r extendL*›
  unfolding *extendL-def wo-rel.adm-woL-def*[*OF wo-rel-r*] **by** *blast*

definition *Extend* :: ‹*'a set* ⇒ *'a set*› **where**
  ‹*Extend S* ≡ ⋃ *n* ∈ *Field r*. *extend S n*›

lemma *extend-subset*: ‹*n* ∈ *Field r* ⟹ *S* ⊆ *extend S n*›
proof (*induct n rule*: *wo-rel.well-order-inductZSL*[*OF wo-rel-r*])
  case *1*
  then show *?case*
    unfolding *extend-def wo-rel.worecZSL-zero*[*OF wo-rel-r adm-woL-extendL*]
    by *simp*
next
  case (*2 i*)
  moreover from *this* **have** ‹*i* ∈ *Field r*›
    by (*meson FieldI1 wo-rel.succ-in wo-rel-r*)
  ultimately show *?case*
    unfolding *extend-def extendS-def*
      wo-rel.worecZSL-succ[*OF wo-rel-r adm-woL-extendL 2*(*1*)] **by** *auto*
next
  case (*3 i*)
  then show *?case*
    unfolding *extend-def extendL-def*

wo-rel.worecZSL-isLim[*OF wo-rel-r adm-woL-extendL 3*(*1*−*2*)]
using *wo-rel-r* **by** (*metis SUP-upper2 emptyE underS-I wo-rel.zero-in-Field wo-rel.zero-smallest*)
qed

lemma *Extend-subset'*: ‹*Field r* ≠ {} ⟹ *S* ⊆ *Extend S*›
unfolding *Extend-def* **using** *extend-subset* **by** *fast*

lemma *extend-underS*: ‹*m* ∈ *underS r n* ⟹ *extend S m* ⊆ *extend S n*›
proof (*induct n rule*: *wo-rel.well-order-inductZSL*[*OF wo-rel-r*])
case *1*
then show *?case*
unfolding *extend-def* **using** *wo-rel-r* **by** (*simp add*: *wo-rel.underS-zero*)
next
case (*2 i*)
moreover from *this* **have** ‹*m* = *i* ∨ *m* ∈ *underS r i*›
by (*metis wo-rel.less-succ underS-E underS-I wo-rel-r*)
ultimately show *?case*
unfolding *extend-def extendS-def*
wo-rel.worecZSL-succ[*OF wo-rel-r adm-woL-extendL 2*(*1*)]
by *auto*
next
case (*3 i*)
then show *?case*
unfolding *extend-def extendL-def*
wo-rel.worecZSL-isLim[*OF wo-rel-r adm-woL-extendL 3*(*1*−*2*)]
by *blast*
qed

lemma *extend-under*: ‹*m* ∈ *under r n* ⟹ *extend S m* ⊆ *extend S n*›
using *extend-underS wo-rel-r*
by (*metis empty-iff in-Above-under set-eq-subset wo-rel.supr-greater wo-rel.supr-under underS-I*
under-Field under-empty)

lemma *consistent-extend*:
assumes ‹*consistent S*›
shows ‹*consistent* (*extend S n*)›
using *assms*
proof (*induct n rule*: *wo-rel.well-order-inductZSL*[*OF wo-rel-r*])
case *1*
then show *?case*
unfolding *extend-def wo-rel.worecZSL-zero*[*OF wo-rel-r adm-woL-extendL*] **.**
next
case (*2 i*)
then show *?case*
unfolding *extend-def extendS-def*
wo-rel.worecZSL-succ[*OF wo-rel-r adm-woL-extendL 2*(*1*)]
by *auto*

```
next
  case (3 i)
  show ?case
  proof (rule ccontr)
    assume ‹¬ consistent (extend S i)›
    then obtain S' where S': ‹finite S'› ‹S' ⊆ (⋃n ∈ underS r i. extend S n)›
‹¬ consistent S'›
      unfolding extend-def extendL-def
        wo-rel.worecZSL-isLim[OF wo-rel-r adm-woL-extendL 3(1−2)]
      using inconsistent-finite by auto
    then obtain ns where ns: ‹S' ⊆ (⋃n ∈ ns. extend S n)› ‹ns ⊆ underS r i›
‹finite ns›
      by (metis finite-subset-Union finite-subset-image)
    moreover have ‹ns ≠ {}›
      using S'(3) assms calculation(1) consistent-hereditary by auto
    ultimately obtain j where ‹∀n ∈ ns. n ∈ under r j› ‹j ∈ underS r i›
      using wo-rel.finite-underS-bound wo-rel-r ns by (meson subset-iff)
    then have ‹∀n ∈ ns. extend S n ⊆ extend S j›
      using extend-under by fast
    then have ‹S' ⊆ extend S j›
      using S' ns(1) by blast
    then show False
      using 3(3) ‹¬ consistent S'› assms consistent-hereditary ‹j ∈ underS r i› by
blast
  qed
qed

lemma consistent-Extend:
  assumes ‹consistent S›
  shows ‹consistent (Extend S)›
  unfolding Extend-def
proof (rule ccontr)
  assume ‹¬ consistent (⋃n ∈ Field r. extend S n)›
  then obtain S' where ‹finite S'› ‹S' ⊆ (⋃n ∈ Field r. extend S n)› ‹¬ consistent
S'›
    using inconsistent-finite by metis
  then obtain m where ‹S' ⊆ (⋃n ∈ under r m. extend S n)› ‹m ∈ Field r›
    using wo-rel.finite-bound-under wo-rel-r
    by (metis SUP-le-iff assms consistent-hereditary emptyE under-empty)
  then have ‹S' ⊆ extend S m›
    using extend-under by fast
  moreover have ‹consistent (extend S m)›
    using assms consistent-extend by blast
  ultimately show False
    using ‹¬ consistent S'› consistent-hereditary by blast
qed

definition maximal' :: ‹'a set ⇒ bool› where
  ‹maximal' S ≡ ∀p ∈ Field r. consistent ({p} ∪ S) ⟶ p ∈ S›
```

```
lemma Extend-bound: ‹n ∈ Field r ⟹ extend S n ⊆ Extend S›
  unfolding Extend-def by blast

lemma maximal'-Extend: ‹maximal' (Extend S)›
  unfolding maximal'-def
proof safe
  fix p
  assume *: ‹p ∈ Field r› ‹consistent ({p} ∪ Extend S)›
  then have ‹{p} ∪ extend S p ⊆ {p} ∪ Extend S›
    unfolding Extend-def by blast
  then have **: ‹consistent ({p} ∪ extend S p)›
    using * consistent-hereditary by blast
  moreover have succ: ‹aboveS r p ≠ {}›
    using * isLimOrd-r wo-rel.isLimOrd-aboveS wo-rel-r by fast
  then have ‹succ r p ∈ Field r›
    using wo-rel-r by (simp add: wo-rel.succ-in-Field)
  moreover have ‹p ∈ extend S (succ r p)›
    using ** unfolding extend-def extendS-def
      wo-rel.worecZSL-succ[OF wo-rel-r adm-woL-extendL succ]
    by simp
  ultimately show ‹p ∈ Extend S›
    using Extend-bound by fast
qed

end

locale MCS =
  fixes consistent :: ‹'a set ⇒ bool›
  assumes infinite-UNIV: ‹infinite (UNIV :: 'a set)›
    and ‹consistent S ⟹ S' ⊆ S ⟹ consistent S'›
    and ‹⋀S. ¬ consistent S ⟹ ∃ S' ⊆ S. finite S' ∧ ¬ consistent S'›

sublocale MCS ⊆ MCS-Lim-Ord ‹|UNIV|›
proof
  show ‹Well-order |UNIV|›
    by simp
next
  have ‹infinite ( Field |UNIV :: 'a set| )›
    using infinite-UNIV by simp
  with card-order-infinite-isLimOrd card-of-Card-order
  show ‹isLimOrd |UNIV :: 'a set|› .
next
  fix S S'
  show ‹consistent S ⟹ S' ⊆ S ⟹ consistent S'›
    using MCS-axioms unfolding MCS-def by blast
next
  fix S S'
  show ‹¬ consistent S ⟹ ∃ S' ⊆ S. finite S' ∧ ¬ consistent S'›
```

using *MCS-axioms* **unfolding** *MCS-def* **by** *blast*
qed

context *MCS* **begin**

lemma *Extend-subset*: ‹$S \subseteq$ *Extend* S›
by (*simp add*: *Extend-subset′*)

definition *maximal* :: ‹*′a set* ⇒ *bool*› **where**
‹*maximal* $S \equiv \forall p.$ *consistent* $(\{p\} \cup S) \longrightarrow p \in S$›

lemma *maximal-maximal′*: ‹*maximal* $S \longleftrightarrow$ *maximal′* S›
unfolding *maximal-def maximal′-def* **by** *simp*

lemma *maximal-Extend*: ‹*maximal* (*Extend* S)›
using *maximal′-Extend maximal-maximal′* **by** *fast*

end

end

theory *Epistemic-Logic* **imports** *Maximal-Consistent-Sets* **begin**

1 Syntax

type-synonym *id* = *string*

datatype *′i fm*
= *FF* (‹⊥›)
| *Pro id*
| *Dis* ‹*′i fm*› ‹*′i fm*› (**infixr** ‹∨› *60*)
| *Con* ‹*′i fm*› ‹*′i fm*› (**infixr** ‹∧› *65*)
| *Imp* ‹*′i fm*› ‹*′i fm*› (**infixr** ‹⟶› *55*)
| *K* *′i* ‹*′i fm*›

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

abbreviation *Neg* (‹¬ -› [*70*] *70*) **where**
‹*Neg p* ≡ *p* ⟶ ⊥›

abbreviation ‹*L i p* ≡ ¬ *K i* (¬ *p*)›

2 Semantics

record (*′i*, *′w*) *frame* =
$\mathcal{W}$ :: ‹*′w set*›

$\mathcal{K}$ :: ‹$'i \Rightarrow 'w \Rightarrow 'w$ *set*›

record $('i, 'w)$ *kripke* =
‹$('i, 'w)$ *frame*› +
π :: ‹$'w \Rightarrow id \Rightarrow bool$›

primrec *semantics* :: ‹$('i, 'w)$ *kripke* $\Rightarrow 'w \Rightarrow 'i$ *fm* $\Rightarrow$ *bool*› (‹-, - $\models$ -› [50, 50, 50] 50) **where**
‹$M, w \models \bot \longleftrightarrow False$›
| ‹$M, w \models Pro\ x \longleftrightarrow \pi\ M\ w\ x$›
| ‹$M, w \models p \lor q \longleftrightarrow M, w \models p \lor M, w \models q$›
| ‹$M, w \models p \land q \longleftrightarrow M, w \models p \land M, w \models q$›
| ‹$M, w \models p \longrightarrow q \longleftrightarrow M, w \models p \longrightarrow M, w \models q$›
| ‹$M, w \models K\ i\ p \longleftrightarrow (\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ M, v \models p)$›

abbreviation *validStar* :: ‹$(('i, 'w)$ *kripke* $\Rightarrow$ *bool*$) \Rightarrow 'i$ *fm set* $\Rightarrow 'i$ *fm* $\Rightarrow$ *bool*›
(‹-; - $\models\star$ -› [50, 50, 50] 50) **where**
‹$P; G \models\star p \equiv \forall M.\ P\ M \longrightarrow$
$(\forall w \in \mathcal{W}\ M.\ (\forall q \in G.\ M, w \models q) \longrightarrow M, w \models p)$›

3 S5 Axioms

definition *reflexive* :: ‹$('i, 'w, 'c)$ *frame-scheme* $\Rightarrow$ *bool*› **where**
‹*reflexive* $M \equiv \forall i.\ \forall w \in \mathcal{W}\ M.\ w \in \mathcal{K}\ M\ i\ w$›

definition *symmetric* :: ‹$('i, 'w, 'c)$ *frame-scheme* $\Rightarrow$ *bool*› **where**
‹*symmetric* $M \equiv \forall i.\ \forall v \in \mathcal{W}\ M.\ \forall w \in \mathcal{W}\ M.\ v \in \mathcal{K}\ M\ i\ w \longleftrightarrow w \in \mathcal{K}\ M\ i\ v$›

definition *transitive* :: ‹$('i, 'w, 'c)$ *frame-scheme* $\Rightarrow$ *bool*› **where**
‹*transitive* $M \equiv \forall i.\ \forall u \in \mathcal{W}\ M.\ \forall v \in \mathcal{W}\ M.\ \forall w \in \mathcal{W}\ M.$
$w \in \mathcal{K}\ M\ i\ v \land u \in \mathcal{K}\ M\ i\ w \longrightarrow u \in \mathcal{K}\ M\ i\ v$›

abbreviation *refltrans* :: ‹$('i, 'w, 'c)$ *frame-scheme* $\Rightarrow$ *bool*› **where**
‹*refltrans* $M \equiv$ *reflexive* $M \land$ *transitive* M›

abbreviation *equivalence* :: ‹$('i, 'w, 'c)$ *frame-scheme* $\Rightarrow$ *bool*› **where**
‹*equivalence* $M \equiv$ *reflexive* $M \land$ *symmetric* $M \land$ *transitive* M›

definition *Euclidean* :: ‹$('i, 'w, 'c)$ *frame-scheme* $\Rightarrow$ *bool*› **where**
‹*Euclidean* $M \equiv \forall i.\ \forall u \in \mathcal{W}\ M.\ \forall v \in \mathcal{W}\ M.\ \forall w \in \mathcal{W}\ M.$
$v \in \mathcal{K}\ M\ i\ u \longrightarrow w \in \mathcal{K}\ M\ i\ u \longrightarrow w \in \mathcal{K}\ M\ i\ v$›

lemma *Imp-intro* [*intro*]: ‹$(M, w \models p \Longrightarrow M, w \models q) \Longrightarrow M, w \models p \longrightarrow q$›
by *simp*

theorem *distribution*: ‹$M, w \models K\ i\ p \land K\ i\ (p \longrightarrow q) \longrightarrow K\ i\ q$›
proof
assume ‹$M, w \models K\ i\ p \land K\ i\ (p \longrightarrow q)$›
then have ‹$M, w \models K\ i\ p$› ‹$M, w \models K\ i\ (p \longrightarrow q)$›

by *simp-all*
then have ‹$\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ M,\ v \models p$› ‹$\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ M,\ v \models p \longrightarrow q$›
by *simp-all*
then have ‹$\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ M,\ v \models q$›
by *simp*
then show ‹$M,\ w \models K\ i\ q$›
by *simp*
qed

theorem *generalization*:
fixes M :: ‹$('i,\ 'w)$ *kripke*›
assumes ‹$\forall (M :: ('i,\ 'w)\ kripke).\ \forall w \in \mathcal{W}\ M.\ M,\ w \models p$› ‹$w \in \mathcal{W}\ M$›
shows ‹$M,\ w \models K\ i\ p$›
proof −
have ‹$\forall w' \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ M,\ w' \models p$›
using *assms* **by** *blast*
then show ‹$M,\ w \models K\ i\ p$›
by *simp*
qed

theorem *truth*:
assumes ‹*reflexive* M› ‹$w \in \mathcal{W}\ M$›
shows ‹$M,\ w \models K\ i\ p \longrightarrow p$›
proof
assume ‹$M,\ w \models K\ i\ p$›
then have ‹$\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ M,\ v \models p$›
by *simp*
moreover have ‹$w \in \mathcal{K}\ M\ i\ w$›
using ‹*reflexive* M› ‹$w \in \mathcal{W}\ M$› **unfolding** *reflexive-def* **by** *blast*
ultimately show ‹$M,\ w \models p$›
using ‹$w \in \mathcal{W}\ M$› **by** *simp*
qed

theorem *pos-introspection*:
assumes ‹*transitive* M› ‹$w \in \mathcal{W}\ M$›
shows ‹$M,\ w \models K\ i\ p \longrightarrow K\ i\ (K\ i\ p)$›
proof
assume ‹$M,\ w \models K\ i\ p$›
then have ‹$\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ M,\ v \models p$›
by *simp*
then have ‹$\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ \forall u \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ v.\ M,\ u \models p$›
using ‹*transitive* M› ‹$w \in \mathcal{W}\ M$› **unfolding** *transitive-def* **by** *blast*
then have ‹$\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ M,\ v \models K\ i\ p$›
by *simp*
then show ‹$M,\ w \models K\ i\ (K\ i\ p)$›
by *simp*
qed

```
theorem neg-introspection:
  assumes ‹symmetric M› ‹transitive M› ‹w ∈ 𝒲 M›
  shows ‹M, w ⊨ ¬ K i p ⟶ K i (¬ K i p)›
proof
  assume ‹M, w ⊨ ¬ (K i p)›
  then obtain u where ‹u ∈ 𝒦 M i w› ‹¬ (M, u ⊨ p)› ‹u ∈ 𝒲 M›
    by auto
  moreover have ‹∀ v ∈ 𝒲 M ∩ 𝒦 M i w. u ∈ 𝒲 M ∩ 𝒦 M i v›
    using ‹u ∈ 𝒦 M i w› ‹symmetric M› ‹transitive M› ‹u ∈ 𝒲 M› ‹w ∈ 𝒲 M›
    unfolding symmetric-def transitive-def by blast
  ultimately have ‹∀ v ∈ 𝒲 M ∩ 𝒦 M i w. M, v ⊨ ¬ K i p›
    by auto
  then show ‹M, w ⊨ K i (¬ K i p)›
    by simp
qed
```

4 Normal Modal Logic

```
primrec eval :: ‹(id ⇒ bool) ⇒ ('i fm ⇒ bool) ⇒ 'i fm ⇒ bool› where
  ‹eval - - ⊥ = False›
| ‹eval g - (Pro x) = g x›
| ‹eval g h (p ∨ q) = (eval g h p ∨ eval g h q)›
| ‹eval g h (p ∧ q) = (eval g h p ∧ eval g h q)›
| ‹eval g h (p ⟶ 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 AK :: ‹('i fm ⇒ bool) ⇒ 'i fm ⇒ bool› (‹- ⊢ -› [50, 50] 50)
  for A :: ‹'i fm ⇒ bool› where
    A1: ‹tautology p ⟹ A ⊢ p›
  | A2: ‹A ⊢ K i p ∧ K i (p ⟶ q) ⟶ K i q›
  | Ax: ‹A p ⟹ A ⊢ p›
  | R1: ‹A ⊢ p ⟹ A ⊢ p ⟶ q ⟹ A ⊢ q›
  | R2: ‹A ⊢ p ⟹ A ⊢ K i p›

primrec imply :: ‹'i fm list ⇒ 'i fm ⇒ 'i fm› (infixr ‹⇝› 56) where
  ‹([] ⇝ q) = q›
| ‹(p # ps ⇝ q) = (p ⟶ ps ⇝ q)›

abbreviation AK-assms (‹-; - ⊢ -› [50, 50, 50] 50) where
  ‹A; G ⊢ p ≡ ∃ qs. set qs ⊆ G ∧ (A ⊢ qs ⇝ p)›
```

5 Soundness

```
lemma eval-semantics:
  ‹eval (pi w) (λq. ⦇𝒲 = W, 𝒦 = r, π = pi⦈, w ⊨ q) p = (⦇𝒲 = W, 𝒦 = r, π
= pi⦈, w ⊨ p)›
```

by (*induct p*) *simp-all*

lemma *tautology*:
assumes ‹*tautology p*›
shows ‹M, $w \models p$›
proof −
from *assms* **have** ‹*eval* (g w) (λq. (|$\mathcal{W}$ = W, $\mathcal{K}$ = r, π = g|), $w \models q$) p› **for** W g r
by *simp*
then have ‹(|$\mathcal{W}$ = W, $\mathcal{K}$ = r, π = g|), $w \models p$› **for** W g r
using *eval-semantics* **by** *fast*
then show ‹M, $w \models p$›
by (*metis kripke.cases*)
qed

theorem *soundness*:
assumes ‹$\bigwedge M\ w\ p.\ A\ p \Longrightarrow P\ M \Longrightarrow w \in \mathcal{W}\ M \Longrightarrow M, w \models p$›
shows ‹$A \vdash p \Longrightarrow P\ M \Longrightarrow w \in \mathcal{W}\ M \Longrightarrow M, w \models p$›
by (*induct p arbitrary*: *w rule*: *AK.induct*) (*auto simp*: *assms tautology*)

6 Derived rules

lemma *K-A2'*: ‹$A \vdash K\ i\ (p \longrightarrow q) \longrightarrow K\ i\ p \longrightarrow K\ i\ q$›
proof −
have ‹$A \vdash K\ i\ p \wedge K\ i\ (p \longrightarrow q) \longrightarrow K\ i\ q$›
using *A2* **by** *fast*
moreover have ‹$A \vdash (P \wedge Q \longrightarrow R) \longrightarrow (Q \longrightarrow P \longrightarrow R)$› **for** P Q R
by (*simp add*: *A1*)
ultimately show *?thesis*
using *R1* **by** *fast*
qed

lemma *K-map*:
assumes ‹$A \vdash p \longrightarrow q$›
shows ‹$A \vdash K\ i\ p \longrightarrow K\ i\ q$›
proof −
note ‹$A \vdash p \longrightarrow q$›
then have ‹$A \vdash K\ i\ (p \longrightarrow q)$›
using *R2* **by** *fast*
moreover have ‹$A \vdash K\ i\ (p \longrightarrow q) \longrightarrow K\ i\ p \longrightarrow K\ i\ q$›
using *K-A2'* **by** *fast*
ultimately show *?thesis*
using *R1* **by** *fast*
qed

lemma *K-LK*: ‹$A \vdash (L\ i\ (\neg\ p) \longrightarrow \neg\ K\ i\ p)$›
proof −
have ‹$A \vdash (p \longrightarrow \neg\ \neg\ p)$›
by (*simp add*: *A1*)

moreover have ‹$A \vdash ((P \longrightarrow Q) \longrightarrow (\neg\ Q \longrightarrow \neg\ P))$› **for** $P\ Q$
  using *A1* **by** *force*
ultimately show *?thesis*
  using *K-map R1* **by** *fast*
qed

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

lemma *K-imply-Cons*:
  assumes ‹$A \vdash ps \rightsquigarrow q$›
  shows ‹$A \vdash p\ \#\ ps \rightsquigarrow q$›
proof –
  have ‹$A \vdash (ps \rightsquigarrow q \longrightarrow p\ \#\ ps \rightsquigarrow q)$›
    by (*simp add*: *A1*)
  with *R1 assms* **show** *?thesis* **.**
qed

lemma *K-right-mp*:
  assumes ‹$A \vdash ps \rightsquigarrow p$› ‹$A \vdash ps \rightsquigarrow (p \longrightarrow q)$›
  shows ‹$A \vdash ps \rightsquigarrow q$›
proof –
  have ‹*tautology* $(ps \rightsquigarrow p \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q)$›
    by (*induct ps*) *simp-all*
  with *A1* **have** ‹$A \vdash ps \rightsquigarrow p \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q$› **.**
  then show *?thesis*
    using *assms R1* **by** *blast*
qed

lemma *tautology-imply-superset*:
  assumes ‹*set ps* $\subseteq$ *set qs*›
  shows ‹*tautology* $(ps \rightsquigarrow r \longrightarrow qs \rightsquigarrow r)$›
proof (*rule ccontr*)
  assume ‹$\neg$ *tautology* $(ps \rightsquigarrow r \longrightarrow qs \rightsquigarrow r)$›
  then obtain *g h* **where** ‹$\neg$ *eval g h* $(ps \rightsquigarrow r \longrightarrow qs \rightsquigarrow r)$›
    by *blast*
  then have ‹*eval g h* $(ps \rightsquigarrow r)$› ‹$\neg$ *eval g h* $(qs \rightsquigarrow r)$›
    by *simp-all*
  then consider (*np*) ‹$\exists\, p \in$ *set ps*. $\neg$ *eval g h p*› | (*r*) ‹$\forall\, p \in$ *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* (*qs* ⇝ *r*)›
by (*induct qs*) *simp-all*
then show *?thesis*
using ‹¬ *eval g h* (*qs* ⇝ *r*)› **by** *blast*
next
case *r*
then have ‹*eval g h* (*qs* ⇝ *r*)›
by (*induct qs*) *simp-all*
then show *?thesis*
using ‹¬ *eval g h* (*qs* ⇝ *r*)› **by** *blast*
qed
qed

lemma *K-imply-weaken*:
assumes ‹*A* ⊢ *ps* ⇝ *q*› ‹*set ps* ⊆ *set ps*′›
shows ‹*A* ⊢ *ps*′ ⇝ *q*›
proof −
have ‹*tautology* (*ps* ⇝ *q* ⟶ *ps*′ ⇝ *q*)›
using ‹*set ps* ⊆ *set ps*′› *tautology-imply-superset* **by** *blast*
then have ‹*A* ⊢ *ps* ⇝ *q* ⟶ *ps*′ ⇝ *q*›
using *A1* **by** *blast*
then show *?thesis*
using ‹*A* ⊢ *ps* ⇝ *q*› *R1* **by** *blast*
qed

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

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

lemma *K-Boole*:
assumes ‹*A* ⊢ (¬ *p*) # *G* ⇝ ⊥›
shows ‹*A* ⊢ *G* ⇝ *p*›
proof −
have ‹*A* ⊢ *G* ⇝ ¬ ¬ *p*›

using *assms K-ImpI* **by** *blast*
moreover have ‹*tautology* ($G \rightsquigarrow \neg\, \neg\, p \longrightarrow G \rightsquigarrow p$)›
by (*induct G*) *simp-all*
then have ‹$A \vdash (G \rightsquigarrow \neg\, \neg\, p \longrightarrow G \rightsquigarrow p)$›
using *A1* **by** *blast*
ultimately show *?thesis*
using *R1* **by** *blast*
qed

lemma *K-DisE*:
assumes ‹$A \vdash p \# G \rightsquigarrow r$› ‹$A \vdash q \# G \rightsquigarrow r$› ‹$A \vdash G \rightsquigarrow p \vee q$›
shows ‹$A \vdash G \rightsquigarrow r$›
proof −
have ‹*tautology* ($p \# G \rightsquigarrow r \longrightarrow q \# G \rightsquigarrow r \longrightarrow G \rightsquigarrow p \vee q \longrightarrow G \rightsquigarrow r$)›
by (*induct G*) *auto*
then have ‹$A \vdash p \# G \rightsquigarrow r \longrightarrow q \# G \rightsquigarrow r \longrightarrow G \rightsquigarrow p \vee q \longrightarrow G \rightsquigarrow r$›
using *A1* **by** *blast*
then show *?thesis*
using *assms R1* **by** *blast*
qed

lemma *K-mp*: ‹$A \vdash p \# (p \longrightarrow q) \# G \rightsquigarrow q$›
by (*meson K-imply-head K-imply-weaken K-right-mp set-subset-Cons*)

lemma *K-swap*:
assumes ‹$A \vdash p \# q \# G \rightsquigarrow r$›
shows ‹$A \vdash q \# p \# G \rightsquigarrow r$›
using *assms K-ImpI* **by** (*metis imply.simps*(*1*−*2*))

lemma *K-DisL*:
assumes ‹$A \vdash p \# ps \rightsquigarrow q$› ‹$A \vdash p' \# ps \rightsquigarrow q$›
shows ‹$A \vdash (p \vee p') \# ps \rightsquigarrow q$›
proof −
have ‹$A \vdash p \# (p \vee p') \# ps \rightsquigarrow q$› ‹$A \vdash p' \# (p \vee p') \# ps \rightsquigarrow q$›
using *assms K-swap K-imply-Cons* **by** *blast+*
moreover have ‹$A \vdash (p \vee p') \# ps \rightsquigarrow p \vee p'$›
using *K-imply-head* **by** *blast*
ultimately show *?thesis*
using *K-DisE* **by** *blast*
qed

lemma *K-distrib-K-imp*:
assumes ‹$A \vdash K\ i\ (G \rightsquigarrow q)$›
shows ‹$A \vdash map\ (K\ i)\ G \rightsquigarrow K\ i\ q$›
proof −
have ‹$A \vdash (K\ i\ (G \rightsquigarrow q) \longrightarrow map\ (K\ i)\ G \rightsquigarrow K\ i\ q)$›
proof (*induct G*)
case *Nil*
then show *?case*

```
      by (simp add: A1)
  next
    case (Cons a G)
    have ‹A ⊢ K i a ∧ K i (a # G ⇝ q) ⟶ K i (G ⇝ q)›
      by (simp add: A2)
    moreover have
      ‹A ⊢ ((K i a ∧ K i (a # G ⇝ q) ⟶ K i (G ⇝ q)) ⟶
        (K i (G ⇝ q) ⟶ map (K i) G ⇝ K i q) ⟶
        (K i a ∧ K i (a # G ⇝ q) ⟶ map (K i) G ⇝ K i q))›
      by (simp add: A1)
    ultimately have ‹A ⊢ K i a ∧ K i (a # G ⇝ q) ⟶ map (K i) G ⇝ K i q›
      using Cons R1 by blast
    moreover have
      ‹A ⊢ ((K i a ∧ K i (a # G ⇝ q) ⟶ map (K i) G ⇝ K i q) ⟶
        (K i (a # G ⇝ q) ⟶ K i a ⟶ map (K i) G ⇝ K i q))›
      by (simp add: A1)
    ultimately have ‹A ⊢ (K i (a # G ⇝ q) ⟶ K i a ⟶ 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

lemma K-trans: ‹A ⊢ (p ⟶ q) ⟶ (q ⟶ r) ⟶ p ⟶ r›
  by (auto intro: A1)

lemma K-L-dual: ‹A ⊢ ¬ L i (¬ p) ⟶ K i p›
proof –
  have ‹A ⊢ K i p ⟶ K i p› ‹A ⊢ ¬ ¬ p ⟶ p›
    by (auto intro: A1)
  then have ‹A ⊢ K i (¬ ¬ p) ⟶ K i p›
    by (auto intro: K-map)
  moreover have ‹A ⊢ (P ⟶ Q) ⟶ (¬ ¬ P ⟶ Q)› for P Q
    by (auto intro: A1)
  ultimately show ‹A ⊢ ¬ ¬ K i (¬ ¬ p) ⟶ K i p›
    by (auto intro: R1)
qed
```

7 Strong Soundness

```
corollary soundness-imply:
  assumes ‹⋀M w p. A p ⟹ P M ⟹ w ∈ 𝒲 M ⟹ M, w ⊨ p›
  shows ‹A ⊢ ps ⇝ p ⟹ P; set ps ⊫⋆ p›
proof (induct ps arbitrary: p)
  case Nil
  then show ?case
```

```
    using soundness[of A P p] assms by simp
next
  case (Cons a ps)
  then show ?case
    using K-ImpI by fastforce
qed

theorem strong-soundness:
  assumes ‹⋀M w p. A p ⟹ P M ⟹ w ∈ 𝒲 M ⟹ M, w ⊨ p›
  shows ‹A; G ⊢ p ⟹ P; G ⊫⋆ p›
proof safe
  fix qs w and M :: ‹('a, 'b) kripke›
  assume ‹A ⊢ qs ⇝ p›
  moreover assume ‹set qs ⊆ G› ‹∀ q ∈ G. M, w ⊨ q›
  then have ‹∀ q ∈ set qs. M, w ⊨ q›
    using ‹set qs ⊆ G› by blast
  moreover assume ‹P M› ‹w ∈ 𝒲 M›
  ultimately show ‹M, w ⊨ p›
    using soundness-imply[of A P qs p] assms by blast
qed
```

8 Completeness

8.1 Consistent sets

```
definition consistent :: ‹('i fm ⇒ bool) ⇒ 'i fm set ⇒ bool› where
  ‹consistent A S ≡ ¬ (A; S ⊢ ⊥)›

lemma inconsistent-subset:
  assumes ‹consistent A V› ‹¬ consistent A ({p} ∪ V)›
  obtains V' where ‹set V' ⊆ V› ‹A ⊢ p # V' ⇝ ⊥›
proof –
  obtain V' where V': ‹set V' ⊆ ({p} ∪ V)› ‹p ∈ set V'› ‹A ⊢ V' ⇝ ⊥›
    using assms unfolding consistent-def by blast
  then have *: ‹A ⊢ p # V' ⇝ ⊥›
    using K-imply-Cons by blast

  let ?S = ‹removeAll p V'›
  have ‹set (p # V') ⊆ set (p # ?S)›
    by auto
  then have ‹A ⊢ p # ?S ⇝ ⊥›
    using * K-imply-weaken by blast
  moreover have ‹set ?S ⊆ V›
    using V'(1) by (metis Diff-subset-conv set-removeAll)
  ultimately show ?thesis
    using that by blast
qed

lemma consistent-consequent:
```

assumes ‹*consistent* A V› ‹$p \in V$› ‹$A \vdash p \longrightarrow q$›
shows ‹*consistent* A ($\{q\} \cup V$)›
proof –
have ‹$\forall V'$. *set* $V' \subseteq V \longrightarrow \neg (A \vdash p \# V' \rightsquigarrow \bot)$›
using ‹*consistent* A V› ‹$p \in V$› **unfolding** *consistent-def*
by (*metis insert-subset list.simps*(*15*))
then have ‹$\forall V'$. *set* $V' \subseteq V \longrightarrow \neg (A \vdash q \# V' \rightsquigarrow \bot)$›
using ‹$A \vdash (p \longrightarrow q)$› *K-imply-head K-right-mp* **by** (*metis imply.simps*(*1*−*2*))
then show *?thesis*
using ‹*consistent* A V› *inconsistent-subset* **by** *metis*
qed

lemma *consistent-consequent*′:
assumes ‹*consistent* A V› ‹$p \in V$› ‹*tautology* ($p \longrightarrow q$)›
shows ‹*consistent* A ($\{q\} \cup V$)›
using *assms consistent-consequent A1* **by** *blast*

lemma *consistent-disjuncts*:
assumes ‹*consistent* A V› ‹$(p \vee q) \in V$›
shows ‹*consistent* A ($\{p\} \cup V$) $\vee$ *consistent* A ($\{q\} \cup V$)›
proof (*rule ccontr*)
assume ‹$\neg$ *?thesis*›
then have ‹$\neg$ *consistent* A ($\{p\} \cup V$)› ‹$\neg$ *consistent* A ($\{q\} \cup V$)›
by *blast*+

then obtain S' T' **where**
S': ‹*set* $S' \subseteq V$› ‹$A \vdash p \# S' \rightsquigarrow \bot$› **and**
T': ‹*set* $T' \subseteq V$› ‹$A \vdash q \# T' \rightsquigarrow \bot$›
using ‹*consistent* A V› *inconsistent-subset* **by** *metis*

from S' **have** p: ‹$A \vdash p \# S' @ T' \rightsquigarrow \bot$›
by (*metis K-imply-weaken Un-upper1 append-Cons set-append*)
moreover from T' **have** q: ‹$A \vdash q \# S' @ T' \rightsquigarrow \bot$›
by (*metis K-imply-head K-right-mp R1 imply.simps*(*2*) *imply-append*)
ultimately have ‹$A \vdash (p \vee q) \# S' @ T' \rightsquigarrow \bot$›
using *K-DisL* **by** *blast*
then have ‹$A \vdash S' @ T' \rightsquigarrow \bot$›
using S'(*1*) T'(*1*) p q ‹*consistent* A V› ‹$(p \vee q) \in V$› **unfolding** *consistent-def*
by (*metis Un-subset-iff insert-subset list.simps*(*15*) *set-append*)
moreover have ‹*set* ($S' @ T'$) $\subseteq V$›
by (*simp add*: S'(*1*) T'(*1*))
ultimately show *False*
using ‹*consistent* A V› **unfolding** *consistent-def* **by** *blast*
qed

lemma *exists-finite-inconsistent*:
assumes ‹$\neg$ *consistent* A ($\{\neg p\} \cup V$)›
obtains W **where** ‹$\{\neg p\} \cup W \subseteq \{\neg p\} \cup V$› ‹$(\neg p) \notin W$› ‹*finite* W› ‹$\neg$ *consistent* A ($\{\neg p\} \cup W$)›

proof −
  obtain *W'* **where** *W'*: ‹*set W'* ⊆ {¬ *p*} ∪ *V*› ‹*A* ⊢ *W'* ⇝ ⊥›
    using *assms* **unfolding** *consistent-def* **by** *blast*
  let *?S* = ‹*removeAll* (¬ *p*) *W'*›
  have ‹¬ *consistent A* ({¬ *p*} ∪ *set ?S*)›
    unfolding *consistent-def* **using** *W'*(*2*) **by** *auto*
  moreover have ‹*finite* (*set ?S*)›
    by *blast*
  moreover have ‹{¬ *p*} ∪ *set ?S* ⊆ {¬ *p*} ∪ *V*›
    using *W'*(*1*) **by** *auto*
  moreover have ‹(¬ *p*) ∉ *set ?S*›
    by *simp*
  ultimately show *?thesis*
    by (*meson that*)
qed

lemma *inconsistent-imply*:
  assumes ‹¬ *consistent A* ({¬ *p*} ∪ *set G*)›
  shows ‹*A* ⊢ *G* ⇝ *p*›
  using *assms K-Boole K-imply-weaken* **unfolding** *consistent-def*
  by (*metis insert-is-Un list.simps*(*15*))

8.2 Maximal consistent sets

lemma *fm-any-size*: ‹∃ *p* :: *'i fm*. *size p* = *n*›
proof (*induct n*)
  case *0*
  then show *?case*
    using *fm.size*(*7*) **by** *blast*
next
  case (*Suc n*)
  then show *?case*
    by (*metis add.commute add-0 add-Suc-right fm.size*(*12*))
qed

lemma *infinite-UNIV-fm*: ‹*infinite* (*UNIV* :: *'i fm set*)›
 using *fm-any-size* **by** (*metis* (*full-types*) *finite-imageI infinite-UNIV-nat surj-def*)

interpretation *MCS* ‹*consistent A*› **for** *A* :: ‹*'i fm* ⇒ *bool*›
proof
  show ‹*infinite* (*UNIV* :: *'i fm set*)›
    using *infinite-UNIV-fm* **.**
next
  fix *S S'*
  assume ‹*consistent A S*› ‹*S'* ⊆ *S*›
  then show ‹*consistent A S'*›
    unfolding *consistent-def* **by** *simp*
next
  fix *S*

```
  assume ‹¬ consistent A S›
  then show ‹∃ S' ⊆ S. finite S' ∧ ¬ consistent A S'›
    unfolding consistent-def by blast
qed

theorem deriv-in-maximal:
  assumes ‹consistent A V› ‹maximal A V› ‹A ⊢ p›
  shows ‹p ∈ V›
  using assms R1 inconsistent-subset unfolding consistent-def maximal-def
  by (metis imply.simps(2))

theorem exactly-one-in-maximal:
  assumes ‹consistent A V› ‹maximal A V›
  shows ‹p ∈ V ⟷ (¬ p) ∉ V›
proof
  assume ‹p ∈ V›
  then show ‹(¬ p) ∉ V›
    using assms K-mp unfolding consistent-def maximal-def
    by (metis empty-subsetI insert-subset list.set(1) list.simps(15))
next
  assume ‹(¬ p) ∉ V›
  have ‹A ⊢ (p ∨ ¬ p)›
    by (simp add: A1)
  then have ‹(p ∨ ¬ p) ∈ V›
    using assms deriv-in-maximal by blast
  then have ‹consistent A ({p} ∪ V) ∨ consistent A ({¬ p} ∪ V)›
    using assms consistent-disjuncts by blast
  then show ‹p ∈ V›
    using ‹maximal A V› ‹(¬ p) ∉ V› unfolding maximal-def by blast
qed

theorem consequent-in-maximal:
  assumes ‹consistent A V› ‹maximal A V› ‹p ∈ V› ‹(p ⟶ q) ∈ V›
  shows ‹q ∈ V›
proof -
  have ‹∀ V'. set V' ⊆ V ⟶ ¬ (A ⊢ p # (p ⟶ q) # V' ⇝ ⊥)›
    using ‹consistent A V› ‹p ∈ V› ‹(p ⟶ q) ∈ V› unfolding consistent-def
    by (metis insert-subset list.simps(15))
  then have ‹∀ V'. set V' ⊆ V ⟶ ¬ (A ⊢ q # V' ⇝ ⊥)›
    by (meson K-mp K-ImpI K-imply-weaken K-right-mp set-subset-Cons)
  then have ‹consistent A ({q} ∪ V)›
    using ‹consistent A V› inconsistent-subset by metis
  then show ?thesis
    using ‹maximal A V› unfolding maximal-def by fast
qed

theorem ax-in-maximal:
  assumes ‹consistent A V› ‹maximal A V› ‹A p›
  shows ‹p ∈ V›
```

using *assms deriv-in-maximal Ax* **by** *blast*

theorem *mcs-properties*:
assumes ‹*consistent A V*› **and** ‹*maximal A V*›
shows ‹$A \vdash p \Longrightarrow p \in V$›
and ‹$p \in V \longleftrightarrow (\neg\, p) \notin V$›
and ‹$p \in V \Longrightarrow (p \longrightarrow q) \in V \Longrightarrow q \in V$›
using *assms deriv-in-maximal exactly-one-in-maximal consequent-in-maximal* **by** *blast+*

lemma *maximal-extension*:
fixes V :: ‹*'i fm set*›
assumes ‹*consistent A V*›
obtains W **where** ‹$V \subseteq W$› ‹*consistent A W*› ‹*maximal A W*›
proof −
let *?W* = ‹*Extend A V*›
have ‹$V \subseteq$ *?W*›
using *Extend-subset* **by** *blast*
moreover have ‹*consistent A ?W*›
using *assms consistent-Extend* **by** *blast*
moreover have ‹*maximal A ?W*›
using *assms maximal-Extend* **by** *blast*
ultimately show *?thesis*
using *that* **by** *blast*
qed

8.3 Canonical model

abbreviation *pi* :: ‹*'i fm set* ⇒ *id* ⇒ *bool*› **where**
‹*pi V x* ≡ *Pro x* ∈ *V*›

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

abbreviation *reach* :: ‹(*'i fm* ⇒ *bool*) ⇒ *'i* ⇒ *'i fm set* ⇒ *'i fm set set*› **where**
‹*reach A i V* ≡ {*W*. *known V i* ⊆ *W*}›

abbreviation *mcss* :: ‹(*'i fm* ⇒ *bool*) ⇒ *'i fm set set*› **where**
‹*mcss A* ≡ {*W*. *consistent A W* ∧ *maximal A W*}›

abbreviation *canonical* :: ‹(*'i fm* ⇒ *bool*) ⇒ (*'i*, *'i fm set*) *kripke*› **where**
‹*canonical A* ≡ ⦇$\mathcal{W}$ = *mcss A*, $\mathcal{K}$ = *reach A*, π = *pi*⦈›

lemma *truth-lemma*:
fixes p :: ‹*'i fm*›
assumes ‹*consistent A V*› **and** ‹*maximal A V*›
shows ‹$p \in V \longleftrightarrow$ *canonical A*, $V \models p$›
using *assms*
proof (*induct p arbitrary*: *V*)

```
  case FF
  then show ?case
  proof safe
    assume ‹⊥ ∈ V›
    then have False
      using ‹consistent A V› K-imply-head unfolding consistent-def
      by (metis bot.extremum insert-subset list.set(1) list.simps(15))
    then show ‹canonical A, V ⊨ ⊥› ..
  next
    assume ‹canonical A, V ⊨ ⊥›
    then show ‹⊥ ∈ V›
      by simp
  qed
next
  case (Pro x)
  then show ?case
    by simp
next
  case (Dis p q)
  then show ?case
  proof safe
    assume ‹(p ∨ q) ∈ V›
    then have ‹consistent A ({p} ∪ V) ∨ consistent A ({q} ∪ V)›
      using ‹consistent A V› consistent-disjuncts by blast
    then have ‹p ∈ V ∨ q ∈ V›
      using ‹maximal A V› unfolding maximal-def by fast
    then show ‹canonical A, V ⊨ (p ∨ q)›
      using Dis by simp
  next
    assume ‹canonical A, V ⊨ (p ∨ q)›
    then consider ‹canonical A, V ⊨ p› | ‹canonical A, V ⊨ q›
      by auto
    then have ‹p ∈ V ∨ q ∈ V›
      using Dis by auto
    moreover have ‹A ⊢ p ⟶ p ∨ q› ‹A ⊢ q ⟶ p ∨ q›
      by (auto simp: A1)
    ultimately show ‹(p ∨ q) ∈ V›
      using Dis.prems deriv-in-maximal consequent-in-maximal by blast
  qed
next
  case (Con p q)
  then show ?case
  proof safe
    assume ‹(p ∧ q) ∈ V›
    then have ‹consistent A ({p} ∪ V)› ‹consistent A ({q} ∪ V)›
      using ‹consistent A V› consistent-consequent' by fastforce+
    then have ‹p ∈ V› ‹q ∈ V›
      using ‹maximal A V› unfolding maximal-def by fast+
    then show ‹canonical A, V ⊨ (p ∧ q)›
```

using *Con* **by** *simp*
next
assume ‹*canonical A*, $V \models (p \wedge q)$›
then have ‹*canonical A*, $V \models p$› ‹*canonical A*, $V \models q$›
by *auto*
then have ‹$p \in V$› ‹$q \in V$›
using *Con* **by** *auto*
moreover have ‹$A \vdash p \longrightarrow q \longrightarrow p \wedge q$›
by (*auto simp*: *A1*)
ultimately show ‹$(p \wedge q) \in V$›
using *Con.prems deriv-in-maximal consequent-in-maximal* **by** *blast*
qed
next
case (*Imp p q*)
then show *?case*
proof *safe*
assume ‹$(p \longrightarrow q) \in V$›
then have ‹*consistent A* $(\{\neg\ p \vee q\} \cup V)$›
using ‹*consistent A V*› *consistent-consequent′* **by** *fastforce*
then have ‹*consistent A* $(\{\neg\ p\} \cup V) \vee$ *consistent A* $(\{q\} \cup V)$›
using ‹*consistent A V*› ‹*maximal A V*› *consistent-disjuncts* **unfolding** *maximal-def* **by** *blast*
then have ‹$(\neg\ p) \in V \vee q \in V$›
using ‹*maximal A V*› **unfolding** *maximal-def* **by** *fast*
then have ‹$p \notin V \vee q \in V$›
using *Imp.prems exactly-one-in-maximal* **by** *blast*
then show ‹*canonical A*, $V \models (p \longrightarrow q)$›
using *Imp* **by** *simp*
next
assume ‹*canonical A*, $V \models (p \longrightarrow q)$›
then consider ‹$\neg$ *canonical A*, $V \models p$› | ‹*canonical A*, $V \models q$›
by *auto*
then have ‹$p \notin V \vee q \in V$›
using *Imp* **by** *auto*
then have ‹$(\neg\ p) \in V \vee q \in V$›
using *Imp.prems exactly-one-in-maximal* **by** *blast*
moreover have ‹$A \vdash \neg\ p \longrightarrow p \longrightarrow q$› ‹$A \vdash q \longrightarrow p \longrightarrow q$›
by (*auto simp*: *A1*)
ultimately show ‹$(p \longrightarrow q) \in V$›
using *Imp.prems deriv-in-maximal consequent-in-maximal* **by** *blast*
qed
next
case (*K i p*)
then show *?case*
proof *safe*
assume ‹$K\ i\ p \in V$›
then show ‹*canonical A*, $V \models K\ i\ p$›
using *K.hyps* **by** *auto*
next

```
    assume ‹canonical A, V ⊨ K i p›

    have ‹¬ consistent A ({¬ p} ∪ known V i)›
    proof
      assume ‹consistent A ({¬ p} ∪ known V i)›
       then obtain W where W: ‹{¬ p} ∪ known V i ⊆ W› ‹consistent A W›
‹maximal A W›
        using ‹consistent A V› maximal-extension by blast
      then have ‹canonical A, W ⊨ ¬ p›
        using K ‹consistent A V› exactly-one-in-maximal by auto
      moreover have ‹W ∈ reach A i V› ‹W ∈ mcss A›
        using W by simp-all
      ultimately have ‹canonical A, V ⊨ ¬ K i p›
        by auto
      then show False
        using ‹canonical A, V ⊨ K i p› by auto
    qed

    then obtain W where W:
      ‹{¬ p} ∪ W ⊆ {¬ p} ∪ known V i› ‹(¬ p) ∉ W› ‹finite W› ‹¬ consistent A
({¬ p} ∪ W)›
      using exists-finite-inconsistent by metis

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

    then have ‹A ⊢ L ⇝ p›
      using W(4) inconsistent-imply by blast
    then have ‹A ⊢ K i (L ⇝ p)›
      using R2 by fast
    then have ‹A ⊢ map (K i) L ⇝ K i p›
      using K-distrib-K-imp by fast
    then have ‹(map (K i) L ⇝ K i p) ∈ V›
      using deriv-in-maximal K.prems(1, 2) by blast
    then show ‹K i p ∈ V›
      using L W(1−2)
    proof (induct L arbitrary: W)
      case (Cons a L)
      then have ‹K i a ∈ V›
        by auto
      then have ‹(map (K i) L ⇝ K i p) ∈ V›
       using Cons(2) ‹consistent A V› ‹maximal A V› consequent-in-maximal by
auto
      then show ?case
        using Cons by auto
    qed simp
  qed
qed
```

```
lemma canonical-model:
  assumes ‹consistent A S› and ‹p ∈ S›
  defines ‹V ≡ Extend A S› and ‹M ≡ canonical A›
  shows ‹M, V ⊨ p› and ‹consistent A V› and ‹maximal A V›
proof −
  have ‹consistent A V›
    using ‹consistent A S› unfolding V-def using consistent-Extend by blast
  have ‹maximal A V›
    unfolding V-def using maximal-Extend by blast
  { fix x
    assume ‹x ∈ S›
    then have ‹x ∈ V›
      unfolding V-def using Extend-subset by blast
    then have ‹M, V ⊨ x›
       unfolding M-def using truth-lemma ‹consistent A V› ‹maximal A V› by
blast }
  then show ‹M, V ⊨ p›
    using ‹p ∈ S› by blast+
  show ‹consistent A V› ‹maximal A V›
    by fact+
qed
```

8.4 Completeness

```
abbreviation valid :: ‹(('i, 'i fm set) kripke ⇒ bool) ⇒ 'i fm set ⇒ 'i fm ⇒ bool›
  (‹-; - ⊫ -› [50, 50, 50] 50)
  where ‹P; G ⊫ p ≡ P; G ⊫⋆ p›

theorem strong-completeness:
  assumes ‹P; G ⊫ p› and ‹P (canonical A)›
  shows ‹A; G ⊢ p›
proof (rule ccontr)
  assume ‹∄ qs. set qs ⊆ G ∧ (A ⊢ qs ⇝ p)›
  then have *: ‹∀ qs. set qs ⊆ G ⟶ ¬ (A ⊢ (¬ p) # qs ⇝ ⊥)›
    using K-Boole by blast

  let ?S = ‹{¬ p} ∪ G›
  let ?V = ‹Extend A ?S›
  let ?M = ‹canonical A›

  have ‹consistent A ?S›
    using * by (metis K-imply-Cons consistent-def inconsistent-subset)
  then have ‹?M, ?V ⊨ (¬ p)› ‹∀ q ∈ G. ?M, ?V ⊨ q›
    using canonical-model by fastforce+
  moreover have ‹?V ∈ mcss A›
    using ‹consistent A ?S› consistent-Extend maximal-Extend by blast
  ultimately have ‹?M, ?V ⊨ p›
    using assms by simp
  then show False
```

using ‹*?M*, *?V* $\models$ (¬ *p*)› **by** *simp*
qed

corollary *completeness*:
assumes ‹*P*; {} $\Vdash$ *p*› **and** ‹*P* (*canonical A*)›
shows ‹*A* ⊢ *p*›
using *assms strong-completeness*[**where** *G*=‹{}›] **by** *simp*

corollary $completeness_A$:
assumes ‹(λ-. *True*); {} $\Vdash$ *p*›
shows ‹*A* ⊢ *p*›
using *assms completeness* **by** *blast*

9 System K

abbreviation *SystemK* (‹- $\vdash_K$ -› [*50*] *50*) **where**
‹$G \vdash_K p \equiv$ (λ-. *False*); *G* ⊢ *p*›

lemma $strong\text{-}soundness_K$: ‹$G \vdash_K p \Longrightarrow P$; *G* $\Vdash\star$ *p*›
using *strong-soundness*[*of* ‹λ-. *False*› ‹λ-. *True*›] **by** *fast*

abbreviation *validK* (‹- $\Vdash_K$ -› [*50*, *50*] *50*) **where**
‹$G \Vdash_K p \equiv$ (λ-. *True*); *G* $\Vdash$ *p*›

lemma $strong\text{-}completeness_K$: ‹$G \Vdash_K p \Longrightarrow G \vdash_K p$›
using *strong-completeness*[*of* ‹λ-. *True*›] **by** *blast*

theorem $main_K$: ‹$G \Vdash_K p \longleftrightarrow G \vdash_K p$›
using $strong\text{-}soundness_K$[*of G p*] $strong\text{-}completeness_K$[*of G p*] **by** *fast*

corollary ‹$G \Vdash_K p \Longrightarrow$ (λ-. *True*); *G* $\Vdash\star$ *p*›
using $strong\text{-}soundness_K$[*of G p*] $strong\text{-}completeness_K$[*of G p*] **by** *fast*

10 System T

Also known as System M

inductive *AxT* :: ‹*'i fm* ⇒ *bool*› **where**
‹*AxT* (*K i p* ⟶ *p*)›

abbreviation *SystemT* (‹- $\vdash_T$ -› [*50*, *50*] *50*) **where**
‹$G \vdash_T p \equiv$ *AxT*; *G* ⊢ *p*›

lemma *soundness-AxT*: ‹*AxT p* ⟹ *reflexive M* ⟹ $w \in \mathcal{W}$ *M* ⟹ *M*, *w* $\models$ *p*›
by (*induct p rule*: *AxT.induct*) (*meson truth*)

lemma $strong\text{-}soundness_T$: ‹$G \vdash_T p \Longrightarrow$ *reflexive*; *G* $\Vdash\star$ *p*›
using *strong-soundness soundness-AxT* **.**

lemma *AxT-reflexive*:
  assumes ‹$AxT \leq A$› **and** ‹*consistent A V*› **and** ‹*maximal A V*›
  shows ‹$V \in$ *reach A i V*›
proof −
  have ‹$(K\ i\ p \longrightarrow p) \in V$› **for** *p*
    using *assms ax-in-maximal AxT.intros* **by** *fast*
  then have ‹$p \in V$› **if** ‹$K\ i\ p \in V$› **for** *p*
    using *that assms consequent-in-maximal* **by** *blast*
  then show *?thesis*
    using *assms* **by** *blast*
qed

lemma $reflexive_T$:
  assumes ‹$AxT \leq A$›
  shows ‹*reflexive* (*canonical A*)›
  unfolding *reflexive-def*
proof *safe*
  fix *i V*
  assume ‹$V \in \mathcal{W}$ (*canonical A*)›
  then have ‹*consistent A V*› ‹*maximal A V*›
    by *simp-all*
  with *AxT-reflexive assms* **have** ‹$V \in$ *reach A i V*› **.**
  then show ‹$V \in \mathcal{K}$ (*canonical A*) *i V*›
    by *simp*
qed

abbreviation *validT* (‹- $\models_T$ -› [*50*, *50*] *50*) **where**
  ‹$G \models_T p \equiv$ *reflexive*; $G \models p$›

lemma $strong\text{-}completeness_T$: ‹$G \models_T p \Longrightarrow G \vdash_T p$›
  using *strong-completeness* $reflexive_T$ **by** *blast*

theorem $main_T$: ‹$G \models_T p \longleftrightarrow G \vdash_T p$›
  using $strong\text{-}soundness_T$[*of G p*] $strong\text{-}completeness_T$[*of G p*] **by** *fast*

corollary ‹$G \models_T p \longrightarrow$ *reflexive*; $G \models\star\ p$›
  using $strong\text{-}soundness_T$[*of G p*] $strong\text{-}completeness_T$[*of G p*] **by** *fast*

11 System KB

inductive *AxB* :: ‹*'i fm* ⇒ *bool*› **where**
  ‹$AxB\ (p \longrightarrow K\ i\ (L\ i\ p))$›

abbreviation *SystemKB* (‹- $\vdash_{KB}$ -› [*50*, *50*] *50*) **where**
  ‹$G \vdash_{KB} p \equiv AxB$; $G \vdash p$›

lemma *soundness-AxB*: ‹$AxB\ p \Longrightarrow$ *symmetric* $M \Longrightarrow w \in \mathcal{W}\ M \Longrightarrow M, w \models p$›
  unfolding *symmetric-def* **by** (*induct p rule*: *AxB.induct*) *auto*

```
lemma strong-soundness_KB: ‹G ⊢_KB p ⟹ symmetric; G ⊫⋆ p›
  using strong-soundness soundness-AxB .

lemma AxB-symmetric':
 assumes ‹AxB ≤ A› ‹consistent A V› ‹maximal A V› ‹consistent A W› ‹maximal
A W›
    and ‹W ∈ reach A i V›
  shows ‹V ∈ reach A i W›
proof -
  have ‹∀ p. K i p ∈ W ⟶ p ∈ V›
  proof (safe, rule ccontr)
    fix p
    assume ‹K i p ∈ W› ‹p ∉ V›
    then have ‹(¬ p) ∈ V›
      using assms(2−3) exactly-one-in-maximal by fast
    then have ‹K i (L i (¬ p)) ∈ V›
      using assms(1−3) ax-in-maximal AxB.intros consequent-in-maximal by fast
    then have ‹L i (¬ p) ∈ W›
      using ‹W ∈ reach A i V› by fast
    then have ‹(¬ K i p) ∈ W›
      using assms(4−5) by (meson K-LK consistent-consequent maximal-def)
    then show False
      using ‹K i p ∈ W› assms(4−5) exactly-one-in-maximal by fast
  qed
  then have ‹known W i ⊆ V›
    by blast
  then show ?thesis
    using assms(2−3) by simp
qed

lemma symmetric_KB:
  assumes ‹AxB ≤ A›
  shows ‹symmetric (canonical A)›
  unfolding symmetric-def
proof (intro allI ballI)
  fix i V W
  assume ‹V ∈ 𝒲 (canonical A)› ‹W ∈ 𝒲 (canonical A)›
  then have ‹consistent A V› ‹maximal A V› ‹consistent A W› ‹maximal A W›
    by simp-all
  with AxB-symmetric' assms have ‹W ∈ reach A i V ⟷ V ∈ reach A i W›
    by metis
  then show ‹(W ∈ 𝒦 (canonical A) i V) = (V ∈ 𝒦 (canonical A) i W)›
    by simp
qed

abbreviation validKB (‹- ⊫_KB -› [50, 50] 50) where
  ‹G ⊫_KB p ≡ symmetric; G ⊫ p›
```

lemma *strong-completeness*$_{KB}$: ‹$G \models_{KB} p \Longrightarrow G \vdash_{KB} p$›
  using *strong-completeness symmetric*$_{KB}$ **by** *blast*

theorem *main*$_{KB}$: ‹$G \models_{KB} p \longleftrightarrow G \vdash_{KB} p$›
  using *strong-soundness*$_{KB}$[*of G p*] *strong-completeness*$_{KB}$[*of G p*] **by** *fast*

corollary ‹$G \models_{KB} p \longrightarrow$ *symmetric*; $G \models\star p$›
  using *strong-soundness*$_{KB}$[*of G p*] *strong-completeness*$_{KB}$[*of G p*] **by** *fast*

12 System K4

inductive *Ax4* :: ‹*'i fm* ⇒ *bool*› **where**
  ‹*Ax4* (*K i p* ⟶ *K i* (*K i p*))›

abbreviation *SystemK4* (‹- $\vdash_{K4}$ -› [*50*, *50*] *50*) **where**
  ‹$G \vdash_{K4} p \equiv$ *Ax4*; $G \vdash p$›

lemma *soundness-Ax4*: ‹*Ax4 p* ⟹ *transitive M* ⟹ *w* ∈ 𝒲 *M* ⟹ *M*, *w* ⊨ *p*›
  by (*induct p rule*: *Ax4.induct*) (*meson pos-introspection*)

lemma *strong-soundness*$_{K4}$: ‹$G \vdash_{K4} p \Longrightarrow$ *transitive*; $G \models\star p$›
  using *strong-soundness soundness-Ax4* **.**

lemma *Ax4-transitive*:
  assumes ‹*Ax4* ≤ *A*› ‹*consistent A V*› ‹*maximal A V*›
    and ‹*W* ∈ *reach A i V*› ‹*U* ∈ *reach A i W*›
  shows ‹*U* ∈ *reach A i V*›
proof −
  have ‹(*K i p* ⟶ *K i* (*K i p*)) ∈ *V*› **for** *p*
    using *assms*(*1*−*3*) *ax-in-maximal Ax4.intros* **by** *fast*
  then have ‹*K i* (*K i p*) ∈ *V*› **if** ‹*K i p* ∈ *V*› **for** *p*
    using *that assms*(*2*−*3*) *consequent-in-maximal* **by** *blast*
  then show *?thesis*
    using *assms*(*4*−*5*) **by** *blast*
qed

lemma *transitive*$_{K4}$:
  assumes ‹*Ax4* ≤ *A*›
  shows ‹*transitive* (*canonical A*)›
  unfolding *transitive-def*
proof *safe*
  fix *i U V W*
  assume ‹*V* ∈ 𝒲 (*canonical A*)›
  then have ‹*consistent A V*› ‹*maximal A V*›
    by *simp-all*
  moreover assume
    ‹*W* ∈ 𝒦 (*canonical A*) *i V*›
    ‹*U* ∈ 𝒦 (*canonical A*) *i W*›
  ultimately have ‹*U* ∈ *reach A i V*›

using *Ax4-transitive assms* **by** *simp*
then show ‹$U \in \mathcal{K}$ (*canonical A*) *i V*›
by *simp*
qed

abbreviation *validK4* (‹- $\models_{K4}$ -› [*50*, *50*] *50*) **where**
‹$G \models_{K4} p \equiv$ *transitive*; $G \models p$›

lemma *strong-completeness*$_{K4}$: ‹$G \models_{K4} p \Longrightarrow G \vdash_{K4} p$›
using *strong-completeness transitive*$_{K4}$ **by** *blast*

theorem *main*$_{K4}$: ‹$G \models_{K4} p \longleftrightarrow G \vdash_{K4} p$›
using *strong-soundness*$_{K4}$[*of G p*] *strong-completeness*$_{K4}$[*of G p*] **by** *fast*

corollary ‹$G \models_{K4} p \longrightarrow$ *transitive*; $G \models\star p$›
using *strong-soundness*$_{K4}$[*of G p*] *strong-completeness*$_{K4}$[*of G p*] **by** *fast*

13 System K5

inductive *Ax5* :: ‹*'i fm* ⇒ *bool*› **where**
‹*Ax5* (*L i p* ⟶ *K i* (*L i p*))›

abbreviation *SystemK5* (‹- $\vdash_{K5}$ -› [*50*, *50*] *50*) **where**
‹$G \vdash_{K5} p \equiv$ *Ax5*; $G \vdash p$›

lemma *soundness-Ax5*: ‹*Ax5 p* ⟹ *Euclidean M* ⟹ $w \in \mathcal{W}$ *M* ⟹ *M*, $w \models p$›
by (*induct p rule*: *Ax5.induct*) (*unfold Euclidean-def semantics.simps*, *blast*)

lemma *strong-soundness*$_{K5}$: ‹$G \vdash_{K5} p \Longrightarrow$ *Euclidean*; $G \models\star p$›
using *strong-soundness soundness-Ax5* **.**

lemma *Ax5-Euclidean*:
assumes ‹*Ax5* ≤ *A*›
‹*consistent A U*› ‹*maximal A U*›
‹*consistent A V*› ‹*maximal A V*›
‹*consistent A W*› ‹*maximal A W*›
and ‹*V* ∈ *reach A i U*› ‹*W* ∈ *reach A i U*›
shows ‹*W* ∈ *reach A i V*›
using *assms*
proof −
{ fix *p*
assume ‹*K i p* ∈ *V*› ‹*p* ∉ *W*›
then have ‹(¬ *p*) ∈ *W*›
using *assms*(*6*−*7*) *exactly-one-in-maximal* **by** *fast*
then have ‹*L i* (¬ *p*) ∈ *U*›
using *assms*(*2*−*3*, *6*−*7*, *9*) *exactly-one-in-maximal* **by** *blast*
then have ‹*K i* (*L i* (¬ *p*)) ∈ *U*›
using *assms*(*1*−*3*) *ax-in-maximal Ax5.intros consequent-in-maximal* **by** *fast*
then have ‹*L i* (¬ *p*) ∈ *V*›

using *assms*(*8*) **by** *blast*
then have ‹¬ K i p ∈ V›
using *assms*(*4*−*5*) *K-LK consequent-in-maximal deriv-in-maximal* **by** *fast*
then have *False*
using *assms*(*4*−*5*) ‹K i p ∈ V› *exactly-one-in-maximal* **by** *fast*
}
then show *?thesis*
by *blast*
qed

lemma *Euclidean*$_{K5}$:
assumes ‹*Ax5* ≤ A›
shows ‹*Euclidean* (*canonical* A)›
unfolding *Euclidean-def*
proof *safe*
fix i U V W
assume ‹$U \in \mathcal{W}$ (*canonical* A)› ‹$V \in \mathcal{W}$ (*canonical* A)› ‹$W \in \mathcal{W}$ (*canonical* A)›
then have
‹*consistent* A U› ‹*maximal* A U›
‹*consistent* A V› ‹*maximal* A V›
‹*consistent* A W› ‹*maximal* A W›
by *simp-all*
moreover assume
‹$V \in \mathcal{K}$ (*canonical* A) i U›
‹$W \in \mathcal{K}$ (*canonical* A) i U›
ultimately have ‹W ∈ *reach* A i V›
using *Ax5-Euclidean assms* **by** *simp*
then show ‹$W \in \mathcal{K}$ (*canonical* A) i V›
by *simp*
qed

abbreviation *validK5* (‹- $\models_{K5}$ -› [*50*, *50*] *50*) **where**
‹$G \models_{K5} p$ ≡ *Euclidean*; $G \models p$›

lemma *strong-completeness*$_{K5}$: ‹$G \models_{K5} p \Longrightarrow G \vdash_{K5} p$›
using *strong-completeness Euclidean*$_{K5}$ **by** *blast*

theorem *main*$_{K5}$: ‹$G \models_{K5} p \longleftrightarrow G \vdash_{K5} p$›
using *strong-soundness*$_{K5}$[*of* G p] *strong-completeness*$_{K5}$[*of* G p] **by** *fast*

corollary ‹$G \models_{K5} p \longrightarrow$ *Euclidean*; $G \models_\star p$›
using *strong-soundness*$_{K5}$[*of* G p] *strong-completeness*$_{K5}$[*of* G p] **by** *fast*

14 System S4

abbreviation *Or* :: ‹($'a \Rightarrow$ *bool*) ⇒ ($'a \Rightarrow$ *bool*) ⇒ $'a \Rightarrow$ *bool*› (**infixl** ‹⊕› *65*)
where
‹$(A \oplus A')\ p \equiv A\ p \lor A'\ p$›

abbreviation *SystemS4* (‹- $\vdash_{S4}$ -› [*50*, *50*] *50*) **where**
‹$G \vdash_{S4} p \equiv AxT \oplus Ax4$; $G \vdash p$›

lemma *soundness-AxT4*: ‹$(AxT \oplus Ax4)\ p \Longrightarrow$ *reflexive* $M \wedge$ *transitive* $M \Longrightarrow w \in \mathcal{W}\ M \Longrightarrow M, w \models p$›
using *soundness-AxT soundness-Ax4* **by** *fast*

lemma *strong-soundness*$_{S4}$: ‹$G \vdash_{S4} p \Longrightarrow$ *refltrans*; $G \models\star\ p$›
using *strong-soundness soundness-AxT4* **.**

abbreviation *validS4* (‹- $\models_{S4}$ -› [*50*, *50*] *50*) **where**
‹$G \models_{S4} p \equiv$ *refltrans*; $G \models p$›

lemma *strong-completeness*$_{S4}$: ‹$G \models_{S4} p \Longrightarrow G \vdash_{S4} p$›
using *strong-completeness*[*of refltrans*] *reflexive*$_T$[*of* ‹$AxT \oplus Ax4$›] *transitive*$_{K4}$[*of* ‹$AxT \oplus Ax4$›]
by *blast*

theorem *main*$_{S4}$: ‹$G \models_{S4} p \longleftrightarrow G \vdash_{S4} p$›
using *strong-soundness*$_{S4}$[*of G p*] *strong-completeness*$_{S4}$[*of G p*] **by** *fast*

corollary ‹$G \models_{S4} p \longrightarrow$ *refltrans*; $G \models\star\ p$›
using *strong-soundness*$_{S4}$[*of G p*] *strong-completeness*$_{S4}$[*of G p*] **by** *fast*

15 System S5

15.1 T + B + 4

abbreviation *SystemS5* (‹- $\vdash_{S5}$ -› [*50*, *50*] *50*) **where**
‹$G \vdash_{S5} p \equiv AxT \oplus AxB \oplus Ax4$; $G \vdash p$›

abbreviation *AxTB4* :: ‹*'i fm* ⇒ *bool*› **where**
‹$AxTB4 \equiv AxT \oplus AxB \oplus Ax4$›

lemma *soundness-AxTB4*: ‹*AxTB4* $p \Longrightarrow$ *equivalence* $M \Longrightarrow w \in \mathcal{W}\ M \Longrightarrow M, w \models p$›
using *soundness-AxT soundness-AxB soundness-Ax4* **by** *fast*

lemma *strong-soundness*$_{S5}$: ‹$G \vdash_{S5} p \Longrightarrow$ *equivalence*; $G \models\star\ p$›
using *strong-soundness soundness-AxTB4* **.**

abbreviation *validS5* (‹- $\models_{S5}$ -› [*50*, *50*] *50*) **where**
‹$G \models_{S5} p \equiv$ *equivalence*; $G \models p$›

lemma *strong-completeness*$_{S5}$: ‹$G \models_{S5} p \Longrightarrow G \vdash_{S5} p$›
using *strong-completeness*[*of equivalence*]
reflexive$_T$[*of AxTB4*] *symmetric*$_{KB}$[*of AxTB4*] *transitive*$_{K4}$[*of AxTB4*]
by *blast*

theorem *main*$_{S5}$: ‹$G \models_{S5} p \longleftrightarrow G \vdash_{S5} p$›
  using *strong-soundness*$_{S5}$[*of G p*] *strong-completeness*$_{S5}$[*of G p*] **by** *fast*

corollary ‹$G \models_{S5} p \longrightarrow$ *equivalence*; $G \models\star p$›
  using *strong-soundness*$_{S5}$[*of G p*] *strong-completeness*$_{S5}$[*of G p*] **by** *fast*

15.2 T + 5

abbreviation *SystemS5′* (‹- $\vdash_{S5}$′′ -› [*50*, *50*] *50*) **where**
  ‹$G \vdash_{S5}' p \equiv AxT \oplus Ax5$; $G \vdash p$›

abbreviation *AxT5* :: ‹′*i fm* ⇒ *bool*› **where**
  ‹$AxT5 \equiv AxT \oplus Ax5$›

lemma *symm-trans-Euclid*: ‹*symmetric M* ⟹ *transitive M* ⟹ *Euclidean M*›
  unfolding *symmetric-def transitive-def Euclidean-def* **by** *blast*

lemma *soundness-AxT5*: ‹*AxT5 p* ⟹ *equivalence M* ⟹ $w \in \mathcal{W}\ M$ ⟹ $M, w \models p$›
  using *soundness-AxT*[*of p M w*] *soundness-Ax5*[*of p M w*] *symm-trans-Euclid*
by *blast*

lemma *strong-soundness*$_{S5}$′: ‹$G \vdash_{S5}' p$ ⟹ *equivalence*; $G \models\star p$›
  using *strong-soundness soundness-AxT5* **.**

lemma *refl-Euclid-equiv*: ‹*reflexive M* ⟹ *Euclidean M* ⟹ *equivalence M*›
  unfolding *reflexive-def symmetric-def transitive-def Euclidean-def* **by** *metis*

lemma *strong-completeness*$_{S5}$′: ‹$G \models_{S5} p \Longrightarrow G \vdash_{S5}' p$›
  using *strong-completeness*[*of equivalence*]
    reflexive$_T$[*of AxT5*] *Euclidean*$_{K5}$[*of AxT5*] *refl-Euclid-equiv* **by** *blast*

theorem *main*$_{S5}$′: ‹$G \models_{S5} p \longleftrightarrow G \vdash_{S5}' p$›
  using *strong-soundness*$_{S5}$′[*of G p*] *strong-completeness*$_{S5}$′[*of G p*] **by** *fast*

15.3 Equivalence between systems

15.3.1 Axiom 5 from B and 4

lemma *K4-L*:
  assumes ‹$Ax4 \leq A$›
  shows ‹$A \vdash L\ i\ (L\ i\ p) \longrightarrow L\ i\ p$›
proof −
  have ‹$A \vdash K\ i\ (\neg\ p) \longrightarrow K\ i\ (K\ i\ (\neg\ p))$›
    using *assms* **by** (*auto intro*: *Ax Ax4.intros*)
  then show *?thesis*
    by (*meson K-LK K-trans R1*)
qed

lemma *KB4-5*:

```
  assumes ‹AxB ≤ A› ‹Ax4 ≤ A›
  shows ‹A ⊢ L i p ⟶ K i (L i p)›
proof -
  have ‹A ⊢ L i p ⟶ K i (L i (L i p))›
    using assms by (auto intro: Ax AxB.intros)
  moreover have ‹A ⊢ L i (L i p) ⟶ L i p›
    using assms by (auto intro: K4-L)
  then have ‹A ⊢ K i (L i (L i p)) ⟶ K i (L i p)›
    using K-map by fast
  ultimately show ?thesis
    using K-trans R1 by metis
qed
```

15.3.2 Axioms B and 4 from T and 5

```
lemma T-L:
  assumes ‹AxT ≤ A›
  shows ‹A ⊢ p ⟶ L i p›
proof -
  have ‹A ⊢ K i (¬ p) ⟶ ¬ p›
    using assms by (auto intro: Ax AxT.intros)
  moreover have ‹A ⊢ (P ⟶ ¬ Q) ⟶ Q ⟶ ¬ P› for P Q
    by (auto intro: A1)
  ultimately show ?thesis
    by (auto intro: R1)
qed

lemma S5'-B:
  assumes ‹AxT ≤ A› ‹Ax5 ≤ A›
  shows ‹A ⊢ p ⟶ K i (L i p)›
proof -
  have ‹A ⊢ L i p ⟶ K i (L i p)›
    using assms(2) by (auto intro: Ax Ax5.intros)
  moreover have ‹A ⊢ p ⟶ L i p›
    using assms(1) by (auto intro: T-L)
  ultimately show ?thesis
    using K-trans R1 by metis
qed

lemma K5-L:
  assumes ‹Ax5 ≤ A›
  shows ‹A ⊢ L i (K i p) ⟶ K i p›
proof -
  have ‹A ⊢ L i (¬ p) ⟶ K i (L i (¬ p))›
    using assms by (auto intro: Ax Ax5.intros)
  then have ‹A ⊢ L i (¬ p) ⟶ K i (¬ K i p)›
    using K-LK by (metis K-map K-trans R1)
  moreover have ‹A ⊢ (P ⟶ Q) ⟶ ¬ Q ⟶ ¬ P› for P Q
    by (auto intro: A1)
```

```
  ultimately have ‹A ⊢ ¬ K i (¬ K i p) ⟶ ¬ L i (¬ p)›
    using R1 by blast
  then have ‹A ⊢ ¬ K i (¬ K i p) ⟶ K i p›
    using K-L-dual R1 K-trans by metis
  then show ?thesis
    by blast
qed

lemma S5'-4:
  assumes ‹AxT ≤ A› ‹Ax5 ≤ A›
  shows ‹A ⊢ K i p ⟶ K i (K i p)›
proof −
  have ‹A ⊢ L i (K i p) ⟶ K i (L i (K i p))›
    using assms(2) by (auto intro: Ax Ax5.intros)
  moreover have ‹A ⊢ K i p ⟶ L i (K i p)›
    using assms(1) by (auto intro: T-L)
  ultimately have ‹A ⊢ K i p ⟶ K i (L i (K i p))›
    using K-trans R1 by metis
  moreover have ‹A ⊢ L i (K i p) ⟶ K i p›
    using assms(2) K5-L by metis
  then have ‹A ⊢ K i (L i (K i p)) ⟶ K i (K i p)›
    using K-map by fast
  ultimately show ?thesis
    using R1 K-trans by metis
qed

lemma S5-S5': ‹AxTB4 ⊢ p ⟹ AxT5 ⊢ p›
proof (induct p rule: AK.induct)
  case (Ax p)
  moreover have ‹AxT5 ⊢ p› if ‹AxT p›
    using that AK.Ax by metis
  moreover have ‹AxT5 ⊢ p› if ‹AxB p›
    using that S5'-B by (metis (no-types, lifting) AxB.cases predicate1I)
  moreover have ‹AxT5 ⊢ p› if ‹Ax4 p›
    using that S5'-4 by (metis (no-types, lifting) Ax4.cases predicate1I)
  ultimately show ?case
    by blast
qed (auto intro: AK.intros)

lemma S5'-S5: ‹AxT5 ⊢ p ⟹ AxTB4 ⊢ p›
proof (induct p rule: AK.induct)
  case (Ax p)
  moreover have ‹AxTB4 ⊢ p› if ‹AxT p›
    using that AK.Ax by metis
  moreover have ‹AxTB4 ⊢ p› if ‹Ax5 p›
    using that KB4-5 by (metis (no-types, lifting) Ax5.cases predicate1I)
  ultimately show ?case
    by blast
qed (auto intro: AK.intros)
```

corollary *S5-S5′-assms*: ‹$G \vdash_{S5} p \longleftrightarrow G \vdash_{S5'} p$›
  using *S5-S5′ S5′-S5* **by** *blast*

16 Acknowledgements

The formalization is inspired by Berghofer's formalization of Henkin-style completeness.

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

end

References

[1] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.

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