Stalnaker's Epistemic Logic

Laura P. Gamboa Guzman

March 19, 2025

Abstract

This work is a formalization of the knowledge fragment Stalnaker's epistemic logic with countably many agents and its soundness and completeness theorems [5, 6, 2], as well as the equivalence between the axiomatization of S4 available in the Epistemic Logic theory and the topological one [1]. It builds on the Epistemic Logic theory, as this includes the formalization of the completeness by canonicity proof for normal modal logics [3, 4].

Contents

theory *Stalnaker-Logic*
  imports *Epistemic-Logic.Epistemic-Logic*

begin

1 Utility

1.1 Some properties of Normal Modal Logics

lemma *duality-taut*: ‹*tautology* $(((K\ i\ p) \longrightarrow K\ i\ (\neg q))\longrightarrow ((L\ i\ q) \longrightarrow (\neg\ K\ i\ p)))$›
  by *force*

lemma *K-imp-trans*:
  assumes ‹$A \vdash (p \longrightarrow q)$› ‹$A \vdash (q \longrightarrow r)$›
  shows ‹$A \vdash (p \longrightarrow r)$›
proof −
  have ‹*tautology* $((p \longrightarrow q) \longrightarrow (\ (q \longrightarrow r) \longrightarrow\ (p \longrightarrow r)))$›
    by *fastforce*
  then show *?thesis*
    by (*meson A1 R1 assms*(*1*) *assms*(*2*))
qed

lemma *K-imp-trans′*:
  assumes ‹$A \vdash (q \longrightarrow r)$›
  shows ‹$A \vdash ((p \longrightarrow q) \longrightarrow (p \longrightarrow r))$›
proof −
  have ‹*tautology* $((q \longrightarrow r) \longrightarrow ((p \longrightarrow q) \longrightarrow (p \longrightarrow r)))$›
    by *fastforce*
  then show *?thesis*
    using *A1 R1 assms* **by** *blast*
qed

lemma *K-imply-multi*:
  assumes ‹$A \vdash (a \longrightarrow b)$› **and** ‹$A \vdash (a \longrightarrow c)$›
  shows ‹$A \vdash (a \longrightarrow (b\wedge c))$›
proof −
  have ‹*tautology* $((a\longrightarrow b)\longrightarrow(a\longrightarrow c)\longrightarrow(a\longrightarrow(b\wedge c)))$›
    by *force*
  then have ‹$A \vdash ((a\longrightarrow b)\longrightarrow(a\longrightarrow c)\longrightarrow(a\longrightarrow(b\wedge c)))$›
    using *A1* **by** *blast*
  then have ‹$A \vdash ((a\longrightarrow c)\longrightarrow(a\longrightarrow(b\wedge c)))$›
    using *assms*(*1*) *R1* **by** *blast*
  then show *?thesis*
    using *assms*(*2*) *R1* **by** *blast*
qed

lemma *K-multi-imply*:
  assumes ‹$A \vdash (a \longrightarrow b \longrightarrow c)$›
  shows ‹$A \vdash ((a\wedge b) \longrightarrow c)$›
proof −
  have ‹*tautology* $((a \longrightarrow b \longrightarrow c) \longrightarrow ((a\wedge b) \longrightarrow c))$›
    by *force*
  then have ‹$A \vdash ((a \longrightarrow b \longrightarrow c) \longrightarrow ((a\wedge b) \longrightarrow c))$›
    using *A1* **by** *blast*

```
  then show ?thesis
    using assms R1 by blast
qed

lemma K-thm: ‹A ⊢ ((K i p) ∧ (L i q) ⟶ L i (p ∧ q))›
proof -
  have ‹tautology (p ⟶ (¬(p ∧ q)) ⟶ ¬ q)›
    by force
  then have ‹A ⊢ (p ⟶ (¬(p ∧ q)) ⟶ ¬ q)›
    by (simp add: A1)
  then have ‹A ⊢ ((K i p) ⟶ K i ((¬(p ∧ q)) ⟶ ¬ q))›
       and ‹A ⊢ (K i ((¬(p ∧ q)) ⟶ ¬ q) ⟶ K i (¬(p ∧ q)) ⟶ K i (¬ q))›
     apply (simp add: K-map)
    by (meson K-A2')
  then have ‹A ⊢ ((K i p) ⟶ K i (¬(p ∧ q)) ⟶ K i (¬ q))›
    using K-imp-trans by blast
  then have ‹A ⊢ ((K i p) ⟶ L i (q) ⟶ L i (p ∧ q))›
    by (metis AK.simps K-imp-trans duality-taut)
  then show ?thesis
    by (simp add: K-multi-imply)
qed

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

lemma imply-conjunct: ‹tautology ((imply G p) ⟶ ((conjunct G) ⟶ p))›
  apply(induction G)
  apply simp
  by force

lemma conjunct-imply: ‹tautology (((conjunct G) ⟶ p) ⟶(imply G p))›
  by (induct G) simp-all

lemma K-imply-conjunct:
  assumes ‹A ⊢ imply G p›
  shows ‹A ⊢ ((conjunct G) ⟶ p)›
  using A1 R1 assms imply-conjunct by blast

lemma K-conjunct-imply:
  assumes ‹A ⊢ ((conjunct G) ⟶ p)›
  shows ‹A ⊢ imply G p›
  using A1 R1 assms conjunct-imply by blast

lemma K-conj-imply-factor:
  fixes A :: ‹('i fm ⇒ bool)›
  shows ‹A ⊢ ((((K i p) ∧ (K i q)) ⟶ r) ⟶((K i (p ∧ q)) ⟶ r))›
proof -
  have *: ‹A ⊢ ((K i (p ∧ q)) ⟶ ((K i p) ∧ (K i q)))›
```

```
  proof (rule ccontr)
    assume ‹¬ A ⊢ ((K i (p ∧ q)) ⟶ ((K i p) ∧ (K i q)))›
    then have ‹consistent A {¬((K i (p ∧ q)) ⟶ ((K i p) ∧ (K i q)))}›
      by (metis imply.simps(1) inconsistent-imply insert-is-Un list.set(1))
    let ?V = ‹Extend A {¬((K i (p ∧ q)) ⟶ ((K i p) ∧ (K i q)))}›
    let ?M = ‹(|𝒲 = mcss A, 𝒦 = reach A, π = pi|)›
    have ‹?V ∈ 𝒲 ?M ∧ ?M, ?V ⊨ ¬((K i (p ∧ q)) ⟶ ((K i p) ∧ (K i q)))›
      using canonical-model ‹consistent A {¬ (K i (p ∧ q) ⟶ K i p ∧ K i q)}›
          insert-iff mem-Collect-eq by fastforce
    then have o: ‹?M, ?V ⊨ ((K i (p ∧ q)) ∧ ¬((K i p) ∧ (K i q)))›
      by auto
    then have ‹?M, ?V ⊨ (K i (p ∧ q))› ‹(?M, ?V ⊨ ¬(K i p)) ∨ (?M, ?V ⊨
¬(K i q))›
      by auto
    then have ‹∀ U ∈ 𝒲 ?M ∩ 𝒦 ?M i ?V. ?M, U ⊨(p ∧ q)›
            ‹∃ U ∈ 𝒲 ?M ∩ 𝒦 ?M i ?V. ?M, U ⊨ ((¬p) ∨ (¬q))›
      using o by auto
    then show False
      by simp
  qed
  then have ‹A ⊢ (((K i (p ∧ q)) ⟶ ((K i p) ∧ (K i q))) ⟶
                ((((K i p) ∧ (K i q)) ⟶ r) ⟶ ((K i (p ∧ q)) ⟶ r)))›
    by (simp add: A1)
  then show ?thesis
    using * R1 by blast
qed

lemma K-conjunction-in: ‹A ⊢ (K i (p ∧ q) ⟶ ((K i p) ∧ K i q))›
proof −
  have o1: ‹A ⊢ ((p∧q) ⟶ p)› and o2: ‹A ⊢ ((p∧q) ⟶ q)›
    apply(simp add: A1)
    by (simp add: A1)
  then have c1: ‹A ⊢ (K i (p ∧ q) ⟶ K i q)› and c2: ‹A ⊢ (K i (p ∧ q) ⟶
K i p)›
    apply (simp add: K-map o2)
    by (simp add: K-map o1)
  then show ?thesis
    by (simp add: K-imply-multi)
qed

lemma K-conjunction-in-mult: ‹A ⊢ ((K i (conjunct G)) ⟶ conjunct (map (K
i) G))›
proof (induct G)
  case Nil
  then show ?case
    by (simp add: A1)
  case (Cons a G)
  then have ‹A ⊢ ((K i (conjunct (a#G))) ⟶ (K i (a ∧ conjunct G)))›
      and ‹A ⊢ ((K i (a ∧ conjunct G)) ⟶ ((K i a) ∧ K i (conjunct G)))›
```

apply (*simp add*: *A1*)
by (*metis K-conjunction-in*)
then have *o1*: ‹*A* ⊢ ((*K i* (*conjunct* (*a*#*G*))) ⟶ ((*K i a*) ∧ *K i* (*conjunct* *G*)))›
using *K-imp-trans* **by** *blast*
then have ‹*A* ⊢ (((*K i a*) ∧ *K i* (*conjunct G*)) ⟶ *K i a*)›
and *o2*: ‹*A* ⊢ (((*K i a*) ∧ *K i* (*conjunct G*)) ⟶ *conjunct* (*map* (*K i*) *G*))›
apply (*simp add*: *A1*)
by (*metis Cons.hyps K-imply-Cons K-multi-imply imply.simps*(*1*) *imply.simps*(*2*))
then have ‹*A* ⊢ (((*K i a*) ∧ *K i* (*conjunct G*)) ⟶ (*K i a*) ∧ *conjunct* (*map* (*K i*) *G*))›
using *K-imply-multi* **by** *blast*
then show *?case*
using *K-imp-trans o1* **by** *auto*
qed

lemma *K-conjunction-out*: ‹*A* ⊢ ((*K i p*) ∧ (*K i q*) ⟶ *K i* (*p* ∧ *q*))›
proof −
have ‹*A* ⊢ (*p* ⟶ *q* ⟶ *p*∧*q*)›
by (*simp add*: *A1*)
then have ‹*A* ⊢ *K i* (*p* ⟶ *q* ⟶ *p*∧*q*)›
by (*simp add*: *R2*)
then have ‹*A* ⊢ ((*K i p*) ⟶ *K i* (*q* ⟶ *p*∧*q*))›
by (*simp add*: *K-map* ‹*A* ⊢ (*p* ⟶ *q* ⟶ *p* ∧ *q*)›)
then have ‹*A* ⊢ ((*K i p*) ⟶ (*K i q*) ⟶ *K i* (*p*∧*q*))›
by (*meson K-A2′ K-imp-trans*)
then show *?thesis*
using *K-multi-imply* **by** *blast*
qed

lemma *K-conjunction-out-mult*: ‹*A* ⊢ (*conjunct* (*map* (*K i*) *G*) ⟶ (*K i* (*conjunct* *G*)))›
proof (*induct G*)
case *Nil*
then show *?case*
by (*metis A1 K-imply-conjunct Nil-is-map-conv R2 conjunct.simps*(*1*) *eval.simps*(*5*) *imply.simps*(*1*))
case (*Cons a G*)
then have ‹*A* ⊢ ((*conjunct* (*map* (*K i*) (*a*#*G*))) ⟶ ((*K i a*) ∧ *conjunct* (*map* (*K i*) *G*)))›
by (*simp add*: *A1*)
then have ∗: ‹*A* ⊢ (((*K i a*) ∧ *conjunct* (*map* (*K i*) *G*))⟶ (*K i a*) ∧ *K i* (*conjunct G*))›
by (*metis Cons.hyps K-imply-Cons K-imply-head K-imply-multi K-multi-imply imply.simps*(*1*) *imply.simps*(*2*))
then have ‹*A* ⊢ (((*K i a*) ∧ *K i* (*conjunct G*))⟶ *K i* (*conjunct* (*a*#*G*)))›
by (*simp add*: *K-conjunction-out*)
then show *?case*
using ∗ *K-imp-trans* **by** *auto*

qed

1.2 More on mcs's properties

lemma *mcs-conjunction*:
assumes ‹*consistent A V*› **and** ‹*maximal A V*›
shows ‹$p \in V \wedge q \in V \longrightarrow (p \wedge q) \in V$›
proof –
have ‹*tautology* $(p \longrightarrow q \longrightarrow (p \wedge q))$›
by *force*
then have ‹$(p \longrightarrow q \longrightarrow (p \wedge q)) \in V$›
using *A1 assms(1) assms(2) deriv-in-maximal* **by** *blast*
then have ‹$p \in V \longrightarrow (q \longrightarrow (p \wedge q)) \in V$›
by (*meson assms(1) assms(2) consequent-in-maximal*)
then show *?thesis*
using *assms(1) assms(2) consequent-in-maximal* **by** *blast*
qed

lemma *mcs-conjunction-mult*:
assumes ‹*consistent A V*› **and** ‹*maximal A V*›
shows ‹(*set* (*S* :: (*'i fm list*)) $\subseteq V \wedge$ *finite* (*set S*)) $\longrightarrow$ (*conjunct S*) $\in V$›
proof(*induct S*)
case *Nil*
then show *?case*
by (*metis K-Boole assms(1) assms(2) conjunct.simps(1) consistent-def inconsistent-subset maximal-def*)
case (*Cons a S*)
then have ‹*set S* $\subseteq$ *set* (*a#S*)›
by (*meson set-subset-Cons*)
then have *c1*: ‹ *set* (*a # S*) $\subseteq V \wedge$ *finite* (*set* (*a # S*)) $\longrightarrow$ *conjunct* (*S*) $\in V \wedge a \in V$›
using *Cons* **by** *fastforce*
then have ‹ *conjunct* (*S*) $\in V \wedge a \in V \longrightarrow$ (*conjunct* (*a#S*)) $\in V$ ›
using *assms(1) assms(2) mcs-conjunction* **by** *auto*
then show *?case*
using *c1* **by** *fastforce*
qed

lemma *reach-dualK*:
assumes ‹*consistent A V*› ‹*maximal A V*›
and ‹*consistent A W*› ‹*maximal A W*› ‹$W \in$ *reach A i V*›
shows ‹$\forall p.\ p \in W \longrightarrow (L\ i\ p) \in V$›
proof (*rule ccontr*)
assume ‹$\neg (\forall p.\ p \in W \longrightarrow (L\ i\ p) \in V)$›
then obtain p' **where** $*$: ‹$p' \in W \wedge (L\ i\ p') \notin V$›
by *presburger*
then have ‹$(\neg L\ i\ p') \in V$›
using *assms(1) assms(2) assms(3) assms(4) assms(5) exactly-one-in-maximal* **by** *blast*

then have ‹K i (¬ p′) ∈ V›
  using *assms(1) assms(2) exactly-one-in-maximal* **by** *blast*
then have ‹(¬ p′) ∈ W›
  using *assms(5)* **by** *blast*
then show *False*
  by (*meson* ∗ *assms(3) assms(4) exactly-one-in-maximal*)
qed

lemma *dual-reach*:
  assumes ‹*consistent A V*› ‹*maximal A V*›
  shows ‹(L i p) ∈ V ⟶ (∃ W. W ∈ reach A i V ∧ p ∈ W)›
proof −
  have ‹(∄ W. W ∈ {W. known V i ⊆ W} ∧ p ∈ W) ⟶ (∀ W. W ∈ {W. known V i ⊆ W} ⟶ (¬p) ∈ W)›
    by *blast*
  then have ‹(∀ W. W ∈ {W. known V i ⊆ W} ⟶ (¬p) ∈ W) ⟶ (∀ W. W ∈ reach A i V ⟶ (¬p) ∈ W)›
    by *fastforce*
  then have ‹(∀ W. W ∈ reach A i V ⟶ (¬p) ∈ W) ⟶ ((K i (¬p)) ∈ V)›
    by *blast*
  then have ‹((K i (¬p)) ∈ V) ⟶ (¬((L i p) ∈ V))›
    using *assms(1) assms(2) exactly-one-in-maximal* **by** *blast*
  then have ‹(∄ W. W ∈ {W. known V i ⊆ W} ∧ p ∈ W) ⟶ ¬((L i p) ∈ V)›
    by *blast*
  then show *?thesis*
    by *blast*
qed

2 Ax.2

definition *weakly-directed* :: ‹('i, 's) kripke ⇒ bool› **where**
  ‹*weakly-directed M* ≡ ∀ i. ∀ s ∈ 𝒲 M. ∀ t ∈ 𝒲 M. ∀ r ∈ 𝒲 M.
    (r ∈ 𝒦 M i s ∧ t ∈ 𝒦 M i s)⟶(∃ u ∈ 𝒲 M. (u ∈ 𝒦 M i r ∧ u ∈ 𝒦 M i t))›

inductive *Ax-2* :: ‹'i fm ⇒ bool› **where**
  ‹*Ax-2* (¬ K i (¬ K i p) ⟶ K i (¬ K i (¬ p)))›

2.1 Soundness

theorem *weakly-directed*:
  assumes ‹*weakly-directed M*› ‹w ∈ 𝒲 M›
  shows ‹M, w ⊨ (L i (K i p) ⟶ K i (L i p))›
proof
  assume ‹M, w ⊨ (L i (K i p))›
  then have ‹∃ v ∈ 𝒲 M ∩ 𝒦 M i w. M, v ⊨ K i p›
    by *simp*
  then have ‹∀ v ∈ 𝒲 M ∩ 𝒦 M i w. ∃ u ∈ 𝒲 M ∩ 𝒦 M i v. M, u ⊨ p›
    using ‹*weakly-directed M*› ‹w ∈ 𝒲 M› **unfolding** *weakly-directed-def*
    by (*metis IntE IntI semantics.simps(6)*)

then have ‹∀ *v* ∈ 𝒲 *M* ∩ 𝒦 *M i w*. *M*, *v* ⊨ *L i p*›
by *simp*
then show ‹*M*, *w* ⊨ *K i* (*L i p*)›
by *simp*
qed

lemma *soundness-Ax-2*: ‹*Ax-2 p* ⟹ *weakly-directed M* ⟹ *w* ∈ 𝒲 *M* ⟹ *M*, *w* ⊨ *p*›
by (*induct p rule*: *Ax-2.induct*) (*meson weakly-directed*)

2.2 Imply completeness

lemma *Ax-2-weakly-directed*:
fixes *A* :: ‹*'i fm* ⇒ *bool*›
assumes ‹∀ *p*. *Ax-2 p* ⟶ *A p*› ‹*consistent A V*› ‹*maximal A V*›
and ‹*consistent A W*› ‹*maximal A W*› ‹*consistent A U*› ‹*maximal A U*›
and ‹*W* ∈ *reach A i V*› ‹*U* ∈ *reach A i V*›
shows ‹∃ *X*. (*consistent A X*) ∧ (*maximal A X*) ∧ *X* ∈ (*reach A i W*) ∩ (*reach A i U*)›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
let *?S* = ‹(*known W i*) ∪ (*known U i*)›
have ‹¬ *consistent A ?S*›
by (*smt* (*verit*, *best*) *Int-Collect* ‹∄ *X*. *consistent A X* ∧ *maximal A X* ∧ *X* ∈ {*Wa*. *known W i* ⊆ *Wa*} ∩ {*W*. *known U i* ⊆ *W*}› *maximal-extension mem-Collect-eq sup.bounded-iff*)
then obtain *S'* **where** ∗: ‹*A* ⊢ *imply S'* ⊥› ‹*set S'* ⊆ *?S*› ‹*finite* (*set S'*)›
unfolding *consistent-def* **by** *blast*
let *?U* = ‹*filter* (λ*p*. *p* ∈ (*known U i*)) *S'*›
let *?W* = ‹*filter* (λ*p*. *p* ∈ (*known W i*)) *S'*›
let *?p* = ‹*conjunct ?U*› **and** *?q* = ‹*conjunct ?W*›
have ‹(*set ?U*) ∪ (*set ?W*) = (*set S'*)›
using ∗ **by** *auto*
then have ‹*A* ⊢ *imply ?U* (*imply ?W* ⊥)›
using *K-imply-weaken imply-append*
by (*metis* (*mono-tags*, *lifting*) ∗(*1*) *set-append subset-refl*)
then have ‹*A* ⊢ (*?p* ⟶ (*imply ?W* ⊥))›
using *K-imply-conjunct* **by** *blast*
then have ‹*tautology* ((*imply ?W* ⊥) ⟶ (*?q* ⟶ ⊥))›
using *imply-conjunct* **by** *blast*
then have ‹*A* ⊢ ((*imply ?W* ⊥) ⟶ (*?q* ⟶ ⊥))›
using *A1* **by** *blast*
then have ‹*A* ⊢ (*?p* ⟶ (*?q* ⟶ ⊥))›
using *K-imp-trans* ‹*A* ⊢ (*conjunct* (*filter* (λ*p*. *p* ∈ *known U i*) *S'*) ⟶ *imply* (*filter* (λ*p*. *p* ∈ *known W i*) *S'*) ⊥)›
by *blast*
then have *o1*: ‹*A* ⊢ ((*?p* ∧ *?q*) ⟶ ⊥)›
by (*meson K-multi-imply*)
moreover have ‹*set ?U* ⊆ (*known U i*)› **and** ‹*set ?W* ⊆ (*known W i*)›

and ‹∀ *p*. *p* ∈ *set ?U* ⟶ (*K i p*) ∈ *U*› **and** ‹∀ *p*. *p* ∈ *set ?W* ⟶ (*K i p*) ∈ *W*›
by *auto*
then have ‹*set* (*map* (*K i*) *?U*) ⊆ *U*› **and** *c1*: ‹*set* (*map* (*K i*) *?W*) ⊆ *W*›
apply (*metis* (*mono-tags*, *lifting*) *imageE set-map subsetI*)
by *auto*
then have *c2*: ‹*conjunct* (*map* (*K i*) *?U*) ∈ *U*› **and** *c2′*: ‹*conjunct* (*map* (*K i*) *?W*) ∈ *W*›
using *assms*(*6*) *assms*(*7*) *mcs-conjunction-mult* **apply** *blast*
using *assms*(*4*) *assms*(*5*) *c1 mcs-conjunction-mult* **by** *blast*
then have ‹((*conjunct* (*map* (*K i*) *?U*)) ⟶ (*K i ?p*)) ∈ *U*›
and *c3*: ‹((*conjunct* (*map* (*K i*) *?W*)) ⟶ (*K i ?q*)) ∈ *W*›
apply (*meson K-conjunction-out-mult assms*(*6*) *assms*(*7*) *deriv-in-maximal*)
by (*meson K-conjunction-out-mult assms*(*4*) *assms*(*5*) *deriv-in-maximal*)
then have *c4*: ‹(*K i ?p*) ∈ *U*› **and** *c4′*: ‹(*K i ?q*) ∈ *W*›
using *assms*(*6*) *assms*(*7*) *c2 consequent-in-maximal* **apply** *blast*
using *assms*(*4*) *assms*(*5*) *c2′ c3 consequent-in-maximal* **by** *blast*
then have ‹(*L i* (*K i ?p*)) ∈ *V*› **and** *c5*: ‹(*L i* (*K i ?q*)) ∈ *V*›
using *assms*(*2*) *assms*(*3*) *assms*(*6*) *assms*(*7*) *assms*(*9*) *exactly-one-in-maximal* **apply** *blast*
using *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *assms*(*8*) *c4′ exactly-one-in-maximal* **by** *blast*
then have ‹(*K i* (*L i ?p*)) ∈ *V*›
by (*meson Ax-2.intros assms*(*1*) *assms*(*2*) *assms*(*3*) *ax-in-maximal consequent-in-maximal*)
then have ‹((*K i* (*L i ?p*)) ∧ (*L i* (*K i ?q*))) ∈ *V*›
using *assms*(*2*) *assms*(*3*) *c5 mcs-conjunction* **by** *blast*
then have ‹(*L i* ((*L i ?p*) ∧ *K i ?q*)) ∈ *V*›
by (*meson K-thm assms*(*2*) *assms*(*3*) *consequent-in-maximal deriv-in-maximal*)
then have ‹(*L i* ((*K i ?q*) ∧ *L i ?p*)) ∈ *V*›
by (*smt* (*verit*) ‹*K i* (*L i* (*conjunct* (*filter* (λ*p*. *p* ∈ *known U i*) *S′*))) ∈ *V*› *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *assms*(*8*) *c4′ exactly-one-in-maximal mcs-conjunction mem-Collect-eq subset-iff*)
then obtain *Z′* **where** *z1*:‹(*consistent A Z′*) ∧ (*maximal A Z′*)› **and** *z2*:‹*Z′* ∈ (*reach A i V*)›
and *z3*: ‹((*K i ?q*) ∧ *L i ?p*) ∈ *Z′*›
using ‹*K i* (*L i* (*conjunct* (*filter* (λ*p*. *p* ∈ *known U i*) *S′*))) ∈ *V*› *assms*(*4*) *assms*(*5*) *assms*(*8*) *c4′ mcs-conjunction* **by** *blast*
then have *z4*: ‹(*L i* (*?q* ∧ *?p*)) ∈ *Z′*›
by (*metis K-thm consequent-in-maximal deriv-in-maximal*)
then have *o2*:‹(*L i* (*L i* (*?q* ∧ *?p*))) ∈ *V*›
using *assms*(*2*) *assms*(*3*) *mcs-properties*(*2*) *z1 z2* **by** *blast*
then have ‹*A* ⊢ *K i* (*K i* (((*?p* ∧ *?q*) ⟶ ⊥)))›
by (*metis R2 o1*)
then have *o3*:‹*K i* (*K i* (((*?p* ∧ *?q*) ⟶ ⊥))) ∈ *V*›
using *assms*(*2*) *assms*(*3*) *deriv-in-maximal* **by** *blast*
then obtain *X1* **where** *x1*:‹(*consistent A X1*) ∧ (*maximal A X1*)› **and** *x2*:‹*X1* ∈ (*reach A i V*)›
and *x3*: ‹(*L i* (*?q* ∧ *?p*)) ∈ *X1*›

using *z1 z2 z4* **by** *blast*
then have *x4*:‹(*K i* (((*?p* ∧ *?q*) ⟶ ⊥))) ∈ *X1*›
using *o3* **by** *blast*
then have *t*:‹∀ *x*. ∀ *y*. *tautology* (((*x*∧*y*)⟶⊥)⟶¬(*y*∧*x*))›
by (*metis eval.simps*(*4*) *eval.simps*(*5*))
then have ‹(((*?p*∧*?q*)⟶⊥)⟶¬(*?q*∧*?p*)) ∈ *X1*›
using *A1 deriv-in-maximal x1* **by** *blast*
then have ‹*K i* (((*?p*∧*?q*)⟶⊥)⟶¬(*?q*∧*?p*)) ∈ *X1*›
by (*meson A1 R2 deriv-in-maximal t x1*)
then have ‹(*K i* ((*?p*∧*?q*)⟶⊥)⟶ *K i* (¬(*?q*∧*?p*))) ∈ *X1*›
by (*meson K-A2′ consequent-in-maximal deriv-in-maximal x1*)
then have ‹(*K i* ((*?p*∧*?q*)⟶⊥)⟶ (¬ *L i*(*?q*∧*?p*))) ∈ *X1*›
using *consequent-in-maximal exactly-one-in-maximal x1 x3 x4* **by** *blast*
then have ‹(¬ *L i*(*?q*∧*?p*)) ∈ *X1* ∧ (*L i*(*?q*∧*?p*)) ∈ *X1*›
using *consequent-in-maximal x1 x4 x3* **by** *blast*
then show *False*
using *exactly-one-in-maximal x1* **by** *blast*
qed

lemma *mcs*$_{-2}$*-weakly-directed*:
fixes *A* :: ‹′*i fm* ⇒ *bool*›
assumes ‹∀ *p*. *Ax-2 p* ⟶ *A p*›
shows ‹*weakly-directed* (|𝒲 = *mcss A*, 𝒦 = *reach A*, π = *pi*|)›
unfolding *weakly-directed-def*
proof (*intro allI ballI*, *auto*)
fix *i V U W*
assume ‹*consistent A V*› ‹*maximal A V*› ‹*consistent A U*› ‹*maximal A U*› ‹*consistent A W*›
‹*maximal A W*› ‹*known V i* ⊆ *U*› ‹*known V i* ⊆ *W*›
then have ‹∃ *X*. (*consistent A X*) ∧ (*maximal A X*) ∧ *X* ∈ (*reach A i W*) ∩ (*reach A i U*)›
using *Ax-2-weakly-directed* [**where** *A*=*A* **and** *V*=*V* **and** *W*=*W* **and** *U*=*U*] *assms IntD2*
by *simp*
then have ‹∃ *X*. (*consistent A X*) ∧ (*maximal A X*) ∧ *X* ∈ (*reach A i W*) ∧ *X* ∈ (*reach A i U*)›
by *simp*
then show ‹∃ *X*. (*consistent A X*) ∧ (*maximal A X*) ∧ *known W i* ⊆ *X* ∧ *known U i* ⊆ *X*›
by *auto*
qed

lemma *imply-completeness-K-2*:
assumes *valid*: ‹∀ (*M* :: (′*i*, ′*i fm set*) *kripke*). ∀ *w* ∈ 𝒲 *M*.
weakly-directed M ⟶ (∀ *q* ∈ *G*. *M*, *w* ⊨ *q*) ⟶ *M*, *w* ⊨ *p*›
shows ‹∃ *qs*. *set qs* ⊆ *G* ∧ (*Ax-2* ⊢ *imply qs p*)›
proof (*rule ccontr*)
assume ‹∄ *qs*. *set qs* ⊆ *G* ∧ *Ax-2* ⊢ *imply qs p*›
then have ∗: ‹∀ *qs*. *set qs* ⊆ *G* ⟶ ¬ *Ax-2* ⊢ *imply* ((¬ *p*) # *qs*) ⊥›

```
    using K-Boole by blast

  let ?S = ‹{¬ p} ∪ G›
  let ?V = ‹Extend Ax-2 ?S›
  let ?M = ‹⦇𝒲 = mcss Ax-2, 𝒦 = reach Ax-2, π = pi⦈›

  have ‹consistent Ax-2 ?S›
    using ∗ by (metis K-imply-Cons consistent-def inconsistent-subset)
  then have ‹?M, ?V ⊨ (¬ p)› ‹∀ q ∈ G. ?M, ?V ⊨ q› ‹consistent Ax-2 ?V›
‹maximal Ax-2 ?V›
    using canonical-model unfolding list-all-def by fastforce+
  moreover have ‹weakly-directed ?M›
    using mcs₋₂-weakly-directed [where A=Ax-2] by fast
  ultimately have ‹?M, ?V ⊨ p›
    using valid by auto
  then show False
    using ‹?M, ?V ⊨ (¬ p)› by simp
qed
```

3 System S4.2

```
abbreviation SystemS4-2 :: ‹'i fm ⇒ bool› (‹⊢S42 -› [50] 50) where
  ‹⊢S42 p ≡ AxT ⊕ Ax4 ⊕ Ax-2 ⊢ p›

abbreviation AxS4-2 :: ‹'i fm ⇒ bool› where
  ‹AxS4-2 ≡ AxT ⊕ Ax4 ⊕ Ax-2›
```

3.1 Soundness

```
abbreviation w-directed-preorder :: ‹('i, 'w) kripke ⇒ bool› where
  ‹w-directed-preorder M ≡ reflexive M ∧ transitive M ∧ weakly-directed M›

lemma soundness-AxS4-2: ‹AxS4-2 p ⟹ w-directed-preorder M ⟹ w ∈ 𝒲 M
⟹ M, w ⊨ p›
  using soundness-AxT soundness-Ax4 soundness-Ax-2 by metis

lemma soundnessS42: ‹⊢S42 p ⟹ w-directed-preorder M ⟹ w ∈ 𝒲 M ⟹ M,
w ⊨ p›
  using soundness soundness-AxS4-2 .
```

3.2 Completeness

```
lemma imply-completeness-S4-2:
  assumes valid: ‹∀ (M :: ('i, 'i fm set) kripke). ∀ w ∈ 𝒲 M.
    w-directed-preorder M ⟶ (∀ q ∈ G. M, w ⊨ q) ⟶ M, w ⊨ p›
  shows ‹∃ qs. set qs ⊆ G ∧ (AxS4-2 ⊢ imply qs p)›
proof (rule ccontr)
  assume ‹∄ qs. set qs ⊆ G ∧ AxS4-2 ⊢ imply qs p›
  then have ∗: ‹∀ qs. set qs ⊆ G ⟶ ¬ AxS4-2 ⊢ imply ((¬ p) # qs) ⊥›
```

using *K-Boole* **by** *blast*

let *?S* = ‹{¬ *p*} ∪ *G*›
let *?V* = ‹*Extend AxS4-2 ?S*›
let *?M* = ‹(|𝒲 = *mcss AxS4-2*, 𝒦 = *reach AxS4-2*, π = *pi*|)›

have ‹*consistent AxS4-2 ?S*›
using ∗ **by** (*metis* (*no-types*, *lifting*) *K-imply-Cons consistent-def inconsistent-subset*)
then have
‹*?M*, *?V* ⊨ (¬ *p*)› ‹∀ *q* ∈ *G*. *?M*, *?V* ⊨ *q*›
‹*consistent AxS4-2 ?V*› ‹*maximal AxS4-2 ?V*›
using *canonical-model* **unfolding** *list-all-def* **by** *fastforce+*
moreover have ‹*w-directed-preorder ?M*›
using $reflexive_T$[*of AxS4-2*] mcs_{-2}*-weakly-directed*[*of AxS4-2*] $transitive_{K4}$[*of AxS4-2*] **by** *auto*
ultimately have ‹*?M*, *?V* ⊨ *p*›
using *valid* **by** *auto*
then show *False*
using ‹*?M*, *?V* ⊨ (¬ *p*)› **by** *simp*
qed

lemma $completeness_{S42}$:
assumes ‹∀ (*M* :: ('*i*, '*i fm set*) *kripke*). ∀ *w* ∈ 𝒲 *M*. *w-directed-preorder M* ⟶ *M*, *w* ⊨ *p*›
shows ‹$\vdash_{S42}$ *p*›
using *assms imply-completeness-S4-2*[**where** *G*=‹{}›] **by** *auto*

abbreviation ‹$valid_{S42}$ *p* ≡ ∀ (*M* :: (*nat*, *nat fm set*) *kripke*). ∀ *w* ∈ 𝒲 *M*. *w-directed-preorder M* ⟶ *M*, *w* ⊨ *p*›

theorem $main_{S42}$: ‹$valid_{S42}$ *p* ⟷ $\vdash_{S42}$ *p*›
using $soundness_{S42}$ $completeness_{S42}$ **by** *fast*

corollary
assumes ‹*w-directed-preorder M*› ‹*w* ∈ 𝒲 *M*›
shows ‹$valid_{S42}$ *p* ⟶ *M*, *w* ⊨ *p*›
using *assms* $soundness_{S42}$ $completeness_{S42}$ **by** *fast*

4 Topological S4 axioms

abbreviation *DoubleImp* (**infixr** ‹⟷› *25*) **where**
‹(*p*⟷*q*) ≡ ((*p* ⟶ *q*) ∧ (*q* ⟶ *p*))›

inductive *System-topoS4* :: ‹'*i fm* ⇒ *bool*› (‹$\vdash_{Top}$ -› [*50*] *50*) **where**
A1': ‹*tautology p* ⟹ $\vdash_{Top}$ *p*›
| *AR*: ‹$\vdash_{Top}$ ((*K i* (*p* ∧ *q*)) ⟷ ((*K i p*) ∧ *K i q*))›
| *AT'*: ‹$\vdash_{Top}$ (*K i p* ⟶ *p*)›
| *A4'*: ‹$\vdash_{Top}$ (*K i p* ⟶ *K i* (*K i p*))›

```
| AN: ‹⊢Top K i ⊤›
| R1': ‹⊢Top p ⟹ ⊢Top (p ⟶ q) ⟹ ⊢Top q›
| RM: ‹⊢Top (p ⟶ q) ⟹ ⊢Top ((K i p) ⟶ K i q)›

lemma topoS4-trans: ‹⊢Top ((p ⟶ q) ⟶ (q ⟶ r) ⟶ p ⟶ r)›
  by (simp add: A1')

lemma topoS4-conjElim: ‹⊢Top (p ∧ q ⟶ q)›
  by (simp add: A1')

lemma topoS4-AxK: ‹⊢Top (K i p ∧ K i (p ⟶ q) ⟶ K i q)›
proof −
  have ‹⊢Top ((p ∧ (p ⟶ q)) ⟶ q)›
    using A1' by force
  then have *: ‹⊢Top (K i (p ∧ (p ⟶ q)) ⟶ K i q)›
    using RM by fastforce
  then have ‹⊢Top (K i p ∧ K i (p ⟶ q) ⟶ K i (p ∧ (p ⟶ q)))›
    using AR topoS4-conjElim System-topoS4.simps by fast
  then show ?thesis
    by (meson * System-topoS4.R1' topoS4-trans)
qed

lemma topoS4-NecR:
  assumes ‹⊢Top p›
  shows‹⊢Top K i p›
proof −
  have ‹⊢Top (⊤ ⟶ p)›
   using assms by (metis System-topoS4.A1' System-topoS4.R1' conjunct.simps(1)
imply.simps(1) imply-conjunct)
  then have ‹⊢Top (K i ⊤ ⟶ K i p)›
    using RM by force
  then show ?thesis
    by (meson AN System-topoS4.R1')
qed

lemma empty-S4: {} ⊢S4 p ⟷ AxT ⊕ Ax4 ⊢ p
  by simp

lemma S4-topoS4: ‹{} ⊢S4 p ⟹ ⊢Top p›
  unfolding empty-S4
proof (induct p rule: AK.induct)
  case (A2 i p q)
  then show ?case using topoS4-AxK .
next
  case (Ax p)
    then show ?case
      using AT' A4' by (metis AxT.cases Ax4.cases)
next
  case (R2 p)
```

```
  then show ?case
    by (simp add: topoS4-NecR)
qed (meson System-topoS4.intros)+

lemma topoS4-S4:
  fixes p :: ‹'i fm›
  shows ‹⊢Top p ⟹ {} ⊢S4 p›
  unfolding empty-S4
proof (induct p rule: System-topoS4.induct)
  case (AT' i p)
  then show ?case
    by (simp add: Ax AxT.intros)
next
  case (A4' i p)
  then show ?case
    by (simp add: Ax Ax4.intros)
next
  case (AR i p q)
    then show ?case
     by (meson K-conj-imply-factor K-conjunction-in K-conjunction-out K-imp-trans'
K-imply-multi R1)
next
  case (AN i)
  then have *: ‹AxT ⊕ Ax4 ⊢ ⊤›
    by (simp add: A1)
  then show ?case
    by (simp add: * R2)
next
  case (RM p q i)
  then have ‹AxT ⊕ Ax4 ⊢ K i (p ⟶ q)›
    by (simp add: R2)
  then show ?case
    by (simp add: K-map RM.hyps(2))
qed (meson AK.intros)+

theorem mainS4': ‹{} ⊫S4 p ⟷ (⊢Top p)›
  using mainS4[of {}] S4-topoS4 topoS4-S4 by fast

end
```

References

[1] M. Aiello. Reasoning About Space: The Modal Way. *Journal of Logic and Computation*, 13(6):889–920, dec 2003.

[2] A. Chagrov, P. Chagrov, and M. Zakharyaschev. *Modal Logic*. Oxford logic guides. Clarendon Press, 1997.

[3] A. H. From. Epistemic logic: Completeness of modal logics. *Archive of Formal Proofs*, Oct. 2018. https://isa-afp.org/entries/Epistemic_Logic.html, Formal proof development.

[4] A. H. From. Formalized soundness and completeness of epistemic logic. pages 1–15, 2021.

[5] R. Stalnaker. On logics of knowledge and belief. *Philosophical Studies*, 128:169–199, 3 2006.

[6] R. Stalnaker. *Lecture Notes | Modal Logic | Linguistics and Philosophy | MIT OpenCourseWare*. 2009.