A meta-modal logic for bisimulations

Alfredo Burrieza Fernando Soler-Toscano
Antonio Yuste-Ginel

July 24, 2025

Abstract

Bisimulations are a fundamental formal tool in the model theory of standard modal logic. Roughly speaking, bisimulations provide a clear answer to a foundational model-theoretical question: Given two (Kripke-style) models, what conditions are sufficient and necessary for them to satisfy the same modal formulas? We propose a modal study of the notion of bisimulation. We extend the basic modal language with a new modality $[b]$, whose intended meaning is universal quantification over all states that are bisimilar to the current one. We provide a sound and complete axiomatisation of the class of all pairs of Kripke models linked by bisimulations.

Contents

This theory introduces a modal logic for reasoning about bisimulations (more details on [1]). The proofs rely on various results concerning maximally consistent sets, drawn from the APF entry *Synthetic Completeness* by Asta Halkjær From [2].

theory *Bisimulation-Logic*
  imports *Synthetic-Completeness.Derivations*
begin

1 Syntax

First, the language $\mathcal{L}_{\Box[b]}$ is introduced:

datatype *'p fm*
  = *Fls* (‹⊥›)
  | *Pro 'p* (‹•›)
  | *Imp* ‹*'p fm*› ‹*'p fm*› (**infixr** ‹⟶› *55*)
  | *Box* ‹*'p fm*› (‹□ -› [*70*] *70*)
  | *FrB* ‹*'p fm*› (‹[*b*] -› [*70*] *70*)

Defined connectives.

abbreviation *Not* (‹¬ -› [*70*] *70*) **where**
  ‹¬ *p* ≡ *p* ⟶ ⊥›
abbreviation *Tru* (‹⊤›) **where**
  ‹⊤ ≡ ¬⊥›
abbreviation *Dis* (**infixr** ‹∨› *60*) **where**
  ‹*A* ∨ *B* ≡ ¬*A* ⟶ *B*›
abbreviation *Con* (**infixr** ‹∧› *65*) **where**
  ‹*A* ∧ *B* ≡ ¬(*A* ⟶ ¬*B*)›
abbreviation *Iff* (**infixr** ‹⟷› *55*) **where**
  ‹*A* ⟷ *B* ≡ (*A* ⟶ *B*) ∧ (*B* ⟶ *A*)›
abbreviation *Dia* (‹◇ -› [*70*] *70*) **where**
  ‹◇*A* ≡ ¬□¬*A*›
abbreviation *FrD* (‹⟨*b*⟩ -› [*70*] *70*) **where**
  ‹⟨*b*⟩*A* ≡ ¬[*b*]¬*A*›

Iteration of modal operators □ and ◇.

primrec *chain-b* :: ‹*nat* ⇒ *'p fm* ⇒ *'p fm*› (‹□^- -› [*70*, *70*] *70*) **where**
  ‹□^*0 f* = *f*›

| ‹□^(Suc n) f = □(□^n f)›

primrec *chain-d* :: ‹*nat* ⇒ *'p fm* ⇒ *'p fm*› (‹◇^- -› [*70*, *70*] *70*) **where**
‹◇^0 *f* = *f*›
| ‹◇^(*Suc n*) *f* = ◇(◇^*n f*)›

lemma *chain-bd-sum*:
‹□^*n* (□^*m F*) = □^(*n*+*m*) *F*› **and**
‹◇^*n* (◇^*m F*) = ◇^(*n*+*m*) *F*›
by (*induct n*) *simp-all*

2 Semantics

This is the type of both left and right models:

datatype (*'p*, *'w*) *model* =
Model (*W*: ‹*'w set*›) (*R*: ‹(*'w* × *'w*) *set*›) (*V*: ‹*'w* ⇒ *'p* ⇒ *bool*›)

Given a model $\mathcal{M}$ *W* $\mathcal{M}$ denotes its set of worlds, *R* $\mathcal{M}$ the accessibility relation and *V* $\mathcal{M}$ the valuation function.

This is the type of a model in $\mathcal{L}_{\Box[b]}$:

datatype (*'p*, *'w*) *modelLb* =
ModelLb (*M1*: ‹(*'p*, *'w*) *model*›) (*M2*: ‹(*'p*, *'w*) *model*›) (*Z* : ‹(*'w* × *'w*) *set*›)

Given a model $\mathfrak{M}$ of $\mathcal{L}_{\Box[b]}$, *M1* $\mathfrak{M}$ denotes the model on the left, *M2* $\mathfrak{M}$ the model on the right and *Z* $\mathfrak{M}$ the bisimulation relation.

Bi-models are a relevant class of $\mathcal{L}_{\Box[b]}$, as we will prove soundness and completeness of the proof system $\vdash_B$ for bi-models. First, the conditions for $\mathfrak{M}$ to be a *bi−model* are introduced.

definition *bi-model* :: ‹(*'p*, *'w*) *modelLb* ⇒ *bool*› **where**
‹*bi-model* $\mathfrak{M}$ ≡
— *M1* and *M2* have non-empty domains
W (*M1* $\mathfrak{M}$) ≠ {} ∧ *W* (*M2* $\mathfrak{M}$) ≠ {} ∧
— *R1* and *R2* are defined in the corresponding domains
R (*M1* $\mathfrak{M}$) ⊆ (*W* (*M1* $\mathfrak{M}$)) × (*W* (*M1* $\mathfrak{M}$)) ∧
R (*M2* $\mathfrak{M}$) ⊆ (*W* (*M2* $\mathfrak{M}$)) × (*W* (*M2* $\mathfrak{M}$)) ∧
— *Z* is a non-empty relation from *W* (*M1* $\mathfrak{M}$) to *W* (*M2* $\mathfrak{M}$)
Z $\mathfrak{M}$ ≠ {} ∧ *Z* $\mathfrak{M}$ ⊆ (*W* (*M1* $\mathfrak{M}$)) × (*W* (*M2* $\mathfrak{M}$)) ∧
— Atomic harmony
(∀ *w w'* . (*w*, *w'*) ∈ *Z* $\mathfrak{M}$ ⟶ ((*V* (*M1* $\mathfrak{M}$)) *w*) = ((*V* (*M2* $\mathfrak{M}$)) *w'*)) ∧
— Forth
(∀ *w w' v* . (*w*, *w'*) ∈ *Z* $\mathfrak{M}$ ∧ (*w*,*v*) ∈ *R* (*M1* $\mathfrak{M}$) ⟶
(∃ *v'* . (*v*,*v'*) ∈ *Z* $\mathfrak{M}$ ∧ (*w'*, *v'*) ∈ *R* (*M2* $\mathfrak{M}$))) ∧
— Back
(∀ *w w' v'* . (*w*, *w'*) ∈ *Z* $\mathfrak{M}$ ∧ (*w'*,*v'*) ∈ *R* (*M2* $\mathfrak{M}$) ⟶ (∃ *v* . (*v*,*v'*) ∈ *Z* $\mathfrak{M}$ ∧ (*w*,*v*) ∈ *R* (*M1* $\mathfrak{M}$)))›

In the semantics, formulas are evaluated differently depending on the pointed world is on the left ($\mathcal{M}$) or on the right ($\mathcal{M}'$). Datatype *ep* (evaluation point) indicates the side of a model in which a given formula is evaluated.

datatype *ep* = m | m′

— Pointed model ($\mathcal{M}$,*w*).
type-synonym (′*p*, ′*w*) *Mctx* = ‹(′*p*, ′*w*) *model* × ′*w*›

— Pointed $\mathcal{L}_{\Box[b]}$-modelLb ($\mathfrak{M}$,$\mathcal{M}^{(\prime)}$,*w*).
type-synonym (′*p*, ′*w*) *MLbCtx* = ‹(′*p*, ′*w*) *modelLb* × *ep* × ′*w*›

Definition of truth in a pointed $\mathcal{L}_{\Box[b]}$-modelLb.

fun *semantics* :: ‹(′*p*, ′*w*) *MLbCtx* ⇒ ′*p fm* ⇒ *bool*› (**infix** ‹⊨$_B$› *50*) **where**
‹- ⊨$_B$(⊥::(′*p fm*)) ⟷ *False*›
| ‹($\mathfrak{M}$, m, *w*) ⊨$_B$ ⋅*P* ⟷ *V* (*M1* $\mathfrak{M}$) *w P*›
| ‹($\mathfrak{M}$, m′, *w*) ⊨$_B$ ⋅*P* ⟷ *V* (*M2* $\mathfrak{M}$) *w P*›
| ‹($\mathfrak{M}$, *e*, *w*) ⊨$_B$ *A* ⟶ *B* ⟷ ($\mathfrak{M}$, *e*, *w*) ⊨$_B$ *A* ⟶ ($\mathfrak{M}$, *e*, *w*) ⊨$_B$ *B*›
| ‹($\mathfrak{M}$, m, *w*) ⊨$_B$ □*A* ⟷ (∀ *v* ∈ *W* (*M1* $\mathfrak{M}$) . (*w*,*v*) ∈ *R* (*M1* $\mathfrak{M}$) ⟶ ($\mathfrak{M}$, m, *v*) ⊨$_B$ *A*)›
| ‹($\mathfrak{M}$, m′, *w*) ⊨$_B$ □*A* ⟷ (∀ *v* ∈ *W* (*M2* $\mathfrak{M}$) . (*w*,*v*) ∈ *R* (*M2* $\mathfrak{M}$) ⟶ ($\mathfrak{M}$, m′, *v*) ⊨$_B$ *A*)›
| ‹($\mathfrak{M}$, m, *w*) ⊨$_B$ [*b*]*A* ⟷ (∀ *w*′ ∈ *W* (*M2* $\mathfrak{M}$) . (*w*,*w*′) ∈ (*Z* $\mathfrak{M}$) ⟶ ($\mathfrak{M}$, m′, *w*′) ⊨$_B$ *A*)›
| ‹($\mathfrak{M}$, m′, *w*) ⊨$_B$ [*b*]*A* ⟷ *True*›

3 Calculus

Function *eval* and *tautology* define what is a propositional tautology.

primrec *eval* :: ‹(′*p* ⇒ *bool*) ⇒ (′*p fm* ⇒ *bool*) ⇒ ′*p fm* ⇒ *bool*› **where**
‹*eval* - - ⊥ = *False*›
| ‹*eval g* - (⋅*P*) = *g P*›
| ‹*eval g h* (*A* ⟶ *B*) = (*eval g h A* ⟶ *eval g h B*)›
| ‹*eval* - *h* (□*A*) = *h* (□*A*)›
| ‹*eval* - *h* ([*b*]*A*) = *h* ([*b*]*A*)›

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

— Example of propositional tautology
lemma ‹*tautology* ([*b*]*A* ∨ ¬[*b*]*A*)› **by** *simp*

Finally, the axiom system ⊢$_B$ is presented.

inductive *Calculus* :: ‹′*p fm* ⇒ *bool*› (‹⊢$_B$ -› [*50*] *50*) **where**
TAU: ‹*tautology A* ⟹ ⊢$_B$ *A*›
| *KSq*: ‹⊢$_B$ □ (*A* ⟶ *B*) ⟶ (□*A* ⟶ □*B*)›
| *Kb*: ‹⊢$_B$ [*b*](*A* ⟶ *B*) ⟶ ([*b*]*A* ⟶[*b*]*B*)›
| *FORTH*: ‹⊢$_B$ (⟨*b*⟩*A* ∧ ◇[*b*]*B*) ⟶ ⟨*b*⟩(*A* ∧ ◇*B*)›
| *BACK*: ‹⊢$_B$ ⟨*b*⟩◇*A* ⟶ ◇⟨*b*⟩*A*›

| *HARM*: ‹$(l = \cdot p \vee l = \neg\cdot p) \Longrightarrow \vdash_B l \longrightarrow [b]l$›
| *NTS*: ‹$\vdash_B [b][b]\bot$›
| *MP*: ‹$\vdash_B A \longrightarrow B \Longrightarrow \vdash_B A \Longrightarrow \vdash_B B$›
| *NSq*: ‹$\vdash_B A \Longrightarrow \vdash_B \Box A$›
| *Nb*: ‹$\vdash_B A \Longrightarrow \vdash_B [b]A$›

Proofs use nested conditionals. Given a list $A = [A_1, ..., A_n]$ of formulas, $A \rightsquigarrow B$ represents $A_1 \longrightarrow (A_2 \longrightarrow ... (A_n \longrightarrow B))$.

primrec *imply* :: ‹*'p fm list* ⇒ *'p fm* ⇒ *'p fm*› (**infixr** ‹⇝› *56*) **where**
‹([] ⇝ *B*) = *B*›
| ‹(*A* # Λ ⇝ *B*) = (*A* ⟶ Λ ⇝ *B*)›

abbreviation *Calculus-assms* (**infix** ‹⊢$_B$› *50*) **where**
‹Λ ⊢$_B$ *A* ≡ ⊢$_B$ Λ ⇝ *A*›

4 Soundness

These lemmas will be used to prove soundness.

lemma *atomic-harm*:
assumes ‹*bi-model* 𝔐›
and (*w*,*w'*)∈ *Z* 𝔐
shows ((*V* (*M1* 𝔐)) *w p*) = ((*V* (*M2* 𝔐)) *w'* *p*) **using** *assms bi-model-def* **by** *metis*

lemma *eval-semantics*:
assumes ‹*bi-model* 𝔐›
shows ‹*eval* (*V* (*M1* 𝔐) *w*) (λ*q*. (𝔐, m, *w*) ⊨$_B$ *q*) *p* = ((𝔐, m, *w*) ⊨$_B$ *p*)› **and**
‹*eval* (*V* (*M2* 𝔐) *w*) (λ*q*. (𝔐, m′, *w*) ⊨$_B$ *q*) *p* = ((𝔐, m′, *w*) ⊨$_B$ *p*)›
by (*induct p*) *simp-all*

Tautologies are always true.

lemma *tautology*:
assumes ‹*tautology A*›
and ‹*bi-model* 𝔐›
shows ‹(𝔐, *e*, *w*) ⊨$_B$ *A*›
by (*metis* (*full-types*) *assms*(*1*,*2*) *ep.exhaust eval-semantics*(*1*,*2*))

Axiom FORTH is valid in all worlds in $\mathcal{M}$ of bi-models.

lemma *b-forth*:
assumes ‹*bi-model* 𝔐› **and**
‹*w* ∈ *W* (*M1* 𝔐)›
shows
‹(𝔐, m, *w*) ⊨$_B$ (⟨*b*⟩*F* ∧ ◇[*b*]*G*) ⟶ ⟨*b*⟩(*F* ∧ ◇*G*)›
proof −
{assume *A*: ‹(𝔐, m, *w*) ⊨$_B$ (⟨*b*⟩*F* ∧ ◇[*b*]*G*)›
then obtain *w'* **where** *w'*:
‹*w'* ∈ *W* (*M2* 𝔐) ∧ (*w*,*w'*) ∈ *Z* 𝔐 ∧ ((𝔐, m′, *w'*) ⊨$_B$ *F*)›

by *fastforce*
obtain *w2* **where** *w2*: ‹*w2* ∈ *W* (*M1* 𝔐) ∧ (*w*, *w2*) ∈ *R* (*M1* 𝔐) ∧
(𝔐, m, *w2*) ⊨$_B$ [*b*]*G*› **using** *A assms* **by** *auto*
then obtain *w2′* **where** *w2′*: ‹*w2′* ∈ *W* (*M2* 𝔐) ∧ (*w′*, *w2′*) ∈ *R* (*M2* 𝔐) ∧
(*w2*, *w2′*) ∈ *Z* 𝔐› **using** *bi-model-def assms w′*
by (*smt* (*verit*, *ccfv-SIG*) *SigmaD2 in-mono*)
hence ‹(𝔐, m, *w*) ⊨$_B$ ⟨*b*⟩(*F* ∧ ◇*G*)› **using** *assms w′ w2* **by** *auto*}
thus *?thesis* **by** *auto*
qed

Axiom FORTH is valid in all worlds on $\mathcal{M}'$ (bi-models).

lemma *b-forth2*:
assumes ‹*bi-model* 𝔐› **and**
‹*w* ∈ *W* (*M2* 𝔐)›
shows
‹(𝔐, m′, *w*) ⊨$_B$ (⟨*b*⟩*F* ∧ ◇[*b*]*G*) ⟶ ⟨*b*⟩(*F* ∧ ◇*G*)›
by *simp*

Axiom BACK is valid in all worlds on $\mathcal{M}$ (bi-models).

lemma *b-back*:
assumes ‹*bi-model* 𝔐› **and**
‹*w* ∈ *W* (*M1* 𝔐)›
shows
‹(𝔐, m, *w*) ⊨$_B$ ⟨*b*⟩◇*F* ⟶ ◇⟨*b*⟩*F*›
proof −
{**assume** ‹(𝔐, m, *w*) ⊨$_B$ ⟨*b*⟩◇*F*›
then obtain *w′* **where** *w′*: ‹*w′* ∈ *W* (*M2* 𝔐) ∧
(*w*,*w′*) ∈ *Z* 𝔐 ∧ (𝔐, m′, *w′*) ⊨$_B$ ◇*F*› **by** *auto*
then obtain *w2′* **where** *w2′*: ‹*w′* ∈ *W* (*M2* 𝔐) ∧
(*w′*,*w2′*) ∈ *R* (*M2* 𝔐) ∧ (𝔐, m′, *w2′*) ⊨$_B$ *F*› **by** *auto*
then obtain *w2* **where** *w2*: ‹*w2* ∈ *W* (*M1* 𝔐) ∧ (*w*,*w2*) ∈ *R* (*M1* 𝔐) ∧
(*w2*,*w2′*) ∈ *Z* 𝔐› **using** *assms bi-model-def w′ w2′*
by (*smt* (*verit*, *best*) *SigmaD2 bi-model-def in-mono*)
hence ‹(𝔐, m, *w2*) ⊨$_B$ ⟨*b*⟩*F*› **using** *assms w2′ w2*
by (*metis* (*no-types*, *lifting*) *SigmaD2 bi-model-def in-mono*
semantics.simps(*1*,*4*,*7*))
hence ‹(𝔐, m, *w*) ⊨$_B$ ◇⟨*b*⟩*F*› **using** *w2 assms* **by** *auto*
}
thus *?thesis* **by** *simp*
qed

Axiom BACK is valid in all worlds on $\mathcal{M}'$ (bi-models).

lemma *b-back2*:
assumes ‹*bi-model* 𝔐› **and**
‹*w* ∈ *W* (*M2* 𝔐)›
shows
‹(𝔐, m′, *w*) ⊨$_B$ ⟨*b*⟩◇*F* ⟶ ◇⟨*b*⟩*F*› **by** *simp*

Soundness theorem

theorem *soundness*:
  ‹ $\vdash_B A \Longrightarrow$ *bi-model* $\mathfrak{M} \Longrightarrow$
    $(w \in W\ (M1\ \mathfrak{M}) \longrightarrow (\mathfrak{M}, \mathrm{m}, w) \models_B A) \wedge$
    $(w \in W\ (M2\ \mathfrak{M}) \longrightarrow (\mathfrak{M}, \mathrm{m}', w) \models_B A)$›
proof (*induct A arbitrary*: *w rule*: *Calculus.induct*)
  case (*TAU F*)
  then show *?case*
    by (*simp add*: *tautology*)
next
  case (*FORTH F G*)
  then show *?case* **by** (*metis b-forth2 b-forth*)
next
  case (*BACK F*)
  then show *?case* **by** (*metis b-back b-back2*)
next
  case (*HARM l*)
  then show *?case* **using** *atomic-harm* **by** *fastforce*
qed *auto*

5 Admissible rules

These lemmas are mostly from the AFP entry "Synthetic Completeness" by Asta Halkjær From.

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

lemma *K-imply-Cons*:
  assumes ‹$A \vdash_B q$›
  shows ‹$p \# A \vdash_B q$›
  using *assms*
  by (*metis K-imply-head MP imply.simps*(*1*,*2*))

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

lemma *deduct1*: ‹$A \vdash_B p \longrightarrow q \Longrightarrow p \# A \vdash_B q$›

by (*meson K-right-mp K-imply-Cons K-imply-head*)

lemma *imply-append* [*iff*]: ‹$(A \ @ \ B \rightsquigarrow r) = (A \rightsquigarrow B \rightsquigarrow r)$›
by (*induct A*) *simp-all*

lemma *imply-swap-append*: ‹$A \ @ \ B \vdash_B r \Longrightarrow B \ @ \ A \vdash_B r$›
proof (*induct B arbitrary*: *A*)
case *Cons*
then show *?case*
by (*metis deduct1 imply.simps*(*2*) *imply-append*)
qed *simp*

lemma *K-ImpI*: ‹$p \ \# \ A \vdash_B q \Longrightarrow A \vdash_B p \longrightarrow q$›
by (*metis imply.simps imply-append imply-swap-append*)

lemma *imply-mem* [*simp*]: ‹$p \in set \ A \Longrightarrow A \vdash_B p$›
using *K-imply-head K-imply-Cons* **by** (*induct A*) *fastforce+*

lemma *add-imply* [*simp*]: ‹$\vdash_B q \Longrightarrow A \vdash_B q$›
using *K-imply-head MP* **by** *auto*

lemma *K-imply-weaken*: ‹$A \vdash_B q \Longrightarrow set \ A \subseteq set \ A' \Longrightarrow A' \vdash_B q$›
by (*induct A arbitrary*: *q*) (*simp*, *metis K-right-mp K-ImpI*
imply-mem insert-subset list.set(*2*))

lemma *K-Boole*:
assumes ‹$(\neg \ p) \ \# \ A \vdash_B \bot$›
shows ‹$A \vdash_B p$›
proof −
have *T*: ‹*tautology* $((\neg \ p \longrightarrow \bot) \longrightarrow \ \neg\neg p)$›
by *simp*
have ‹$A \vdash_B \neg \ \neg \ p$›
using *assms K-ImpI T*
using *TAU K-right-mp add-imply* **by** *blast*
moreover have ‹*tautology* $(A \rightsquigarrow \neg \ \neg \ p \longrightarrow A \rightsquigarrow p)$›
by (*induct A*) *simp-all*
then have ‹$\vdash_B (A \rightsquigarrow \neg \ \neg \ p \longrightarrow A \rightsquigarrow p)$›
using *TAU* **by** *blast*
ultimately show *?thesis*
using *MP* **by** *blast*
qed

lemma *MP-chain*:
assumes ‹$\vdash_B A \longrightarrow B$›
and ‹$\vdash_B B \longrightarrow C$›
shows ‹$\vdash_B A \longrightarrow C$›
proof −
have ‹$\vdash_B (A \longrightarrow B) \longrightarrow ((\ B \longrightarrow C) \longrightarrow (A \longrightarrow C))$› **using** *TAU* **by** *force*
thus *?thesis* **using** *assms MP* **by** *auto*

qed

This locale is use to prove common results of normal modal operators. As both $\Box$ and $[b]$ are normal, result involving **K** in *Kop* will be applied to them.

locale *Kop* =
  fixes *K* :: *'p fm* ⇒ *'p fm* (‹**K** -› [*70*] *70*)
  assumes *Kax*: ⊢$_B$ **K** (*A* ⟶ *B*) ⟶ (**K** *A* ⟶ **K** *B*)
    and *KN*: ⊢$_B$ *A* ⟹ ⊢$_B$ **K** *A*

context *Kop* **begin**

abbreviation *P* (‹**P** -› [*70*] *70*) **where** ‹**P** *A* ≡ ¬**K**¬*A*›

lemma *K-distrib-K-imp*:
  assumes ‹⊢$_B$ **K** (*A* ⇝ *q*)›
  shows ‹*map* (λ *x* . **K** *x*) *A* ⊢$_B$ **K** *q*›
proof −
  have ‹⊢$_B$ **K** (*A* ⇝ *q*) ⟶ *map* (λ *x* . **K** *x*) *A* ⇝ **K** *q*›
  proof (*induct A*)
    case *Nil*
    then show *?case*
      by (*simp add*: *TAU*)
  next
    case (*Cons a A*)
    have ‹⊢$_B$ **K** (*a* # *A* ⇝ *q*) ⟶ **K** *a* ⟶ **K** (*A* ⇝ *q*)›
      by (*simp add*: *Kax*)
    moreover have
      ‹⊢$_B$ ((**K** (*a* # *A* ⇝ *q*) ⟶ **K** *a* ⟶ **K** (*A* ⇝ *q*)) ⟶
        (**K** (*A* ⇝ *q*) ⟶ *map* (λ *x* . **K** *x*) *A* ⇝ **K** *q*) ⟶
        (**K** (*a* # *A* ⇝ *q*) ⟶ **K** *a* ⟶ *map* (λ *x* . **K** *x*) *A* ⇝ **K** *q*))›
      using *TAU* **by** *force*
    ultimately have ‹⊢$_B$ **K** (*a* # *A* ⇝ *q*) ⟶ **K** *a* ⟶ *map* (λ *x* . **K** *x*) *A* ⇝
K *q*›
      using *Cons MP* **by** *blast*
    then show *?case*
      by *simp*
  qed
  then show *?thesis*
    using *assms MP* **by** *blast*
qed

lemma *Kpos*:
  shows ‹⊢$_B$ **K**(*A* ⟶ *B*) ⟶ (**P***A* ⟶ **P***B*)›
proof−
  have ‹⊢$_B$ (*A* ⟶ *B*) ⟶ (¬*B* ⟶ ¬*A*)› **using** *TAU* **by** *force*
  hence *1*: ‹⊢$_B$ **K**(*A* ⟶ *B*) ⟶ (**K**¬*B* ⟶ **K**¬*A*)›
   by (*meson KN Kax MP MP-chain*)
  have ‹⊢$_B$ (**K**¬*B* ⟶ **K**¬*A*) ⟶ (**P***A* ⟶ **P***B*)› **using** *TAU* **by** *force*

thus *?thesis* **using** *1 MP-chain* **by** *blast*
qed

end

Both $\Box$ and $[b]$ are normal modal operators.

interpretation *KBox*: *Kop* λ *A* . $\Box$ *A*
by (*simp add*: *KSq Kop-def NSq*)

interpretation *KFrB*: *Kop* λ *A* . $[b]$ *A*
by (*simp add*: *Kb Kop.intro Nb*)

Some other useful theorems of $\vdash_B$ that are used in later proofs.

First, the box-version of BACK axiom.

lemma *BACK-rev*:
$\langle \vdash_B \Box[b]F \longrightarrow [b]\Box F \rangle$
proof −
have *A*: $\langle \vdash_B (\neg[b]\neg\neg\Box\neg\neg F \longrightarrow \neg\ \Box\neg\neg[b]\neg\neg F) \rangle$ **using** *BACK* **.**
have $\langle \vdash_B (\neg[b]\neg\neg\Box\neg\neg F \longrightarrow \neg\ \Box\neg\neg[b]\neg\neg F) \longrightarrow (\ \Box\neg\neg[b]\neg\neg F \longrightarrow [b]\neg\neg\Box\neg\neg F) \rangle$

using *TAU MP* **by** *force*
hence *f1*: $\langle \vdash_B \Box\neg\neg[b]\neg\neg F \longrightarrow [b]\neg\neg\Box\neg\neg F \rangle$ **using** *A MP* **by** *auto*
have $\langle \vdash_B\ \Box([b]\neg\neg F \longrightarrow \neg\neg[b]\neg\neg F) \rangle$ **using** *TAU NSq* **by** *force*
hence *f2*: $\langle \vdash_B\ \Box[b]\neg\neg F \longrightarrow \Box\neg\neg[b]\neg\neg F \rangle$ **using** *KSq MP* **by** *auto*
have $\langle \vdash_B [b](F \longrightarrow \neg\neg F) \rangle$ **using** *TAU Nb* **by** *force*
hence $\langle \vdash_B \Box([b]F \longrightarrow [b]\neg\neg F) \rangle$ **using** *Kb NSq MP* **by** *force*
hence $\langle \vdash_B \Box[b]F \longrightarrow \Box[b]\neg\neg F \rangle$ **using** *KSq MP* **by** *auto*
hence *f3*: $\langle \vdash_B \Box[b]F \longrightarrow [b]\neg\neg\Box\neg\neg F \rangle$ **using** *f1 f2 MP-chain* **by** *blast*
have $\langle \vdash_B [b](\neg\neg\Box\neg\neg F \longrightarrow \Box\neg\neg F) \rangle$ **using** *TAU Nb* **by** *force*
hence *f4*: $\langle \vdash_B [b]\neg\neg\Box\neg\neg F \longrightarrow [b]\Box\neg\neg F \rangle$ **using** *Kb MP* **by** *auto*
have $\langle \vdash_B [b]\Box(\neg\neg F \longrightarrow F) \rangle$ **using** *TAU Nb NSq* **by** *force*
hence $\langle \vdash_B [b]\Box\neg\neg F \longrightarrow [b]\Box F \rangle$ **using** *KSq Kb MP Nb* **by** *metis*
thus *?thesis* **using** *f3 f4 MP-chain* **by** *blast*
qed

— Generalization of axiom NTS.
lemma *NTSgen*:
$\langle \vdash_B [b]\ \Box\widehat{}n\ [b]\bot \rangle$
proof (*induct n*)
case *0*
then show *?case* **using** *NTS chain-b-def* **by** *simp*
next
case (*Suc n*)
fix *n*
assume $\langle \vdash_B [b]\ \Box\widehat{}n\ [b]\ (\bot::'p\ fm) \rangle$
hence $\langle \vdash_B\ \Box\ [b]\ \Box\widehat{}n\ [b]\ (\bot::'p\ fm) \rangle$ **using** *NSq* **by** *auto*
thus $\langle \vdash_B\ [b]\ \Box\widehat{}(Suc\ n)\ [b]\ (\bot::'p\ fm) \rangle$ **using** *BACK-rev MP* **by** *auto*
qed

6 Maximal Consistent Sets

These definitions and lemmas are mostly from the AFP entry "Synthetic Completeness" by Asta Halkjær From.

definition *consistent* :: ‹*'p fm set* ⇒ *bool*› **where**
‹*consistent S* ≡ ∀ *A*. *set A* ⊆ *S* ⟶ ¬ *A* ⊢$_B$ ⊥›

interpretation *MCS-No-Witness-UNIV consistent*
proof
show ‹*infinite* (*UNIV* :: *'p fm set*)›
using *infinite-UNIV-size*[*of* ‹λ*p*. *p* ⟶ *p*›] **by** *simp*
qed (*auto simp*: *consistent-def*)

interpretation *Derivations-Cut-MCS consistent Calculus-assms*
proof
fix *A B* **and** *p* :: ‹*'p fm*›
assume ‹⊢$_B$ *A* ⇝ *p*› ‹*set A* = *set B*›
then show ‹⊢$_B$ *B* ⇝ *p*›
using *K-imply-weaken* **by** *blast*
next
fix *S* :: ‹*'p fm set*›
show ‹*consistent S* = (∀ *A*. *set A* ⊆ *S* ⟶ (∃ *q*. ¬ *A* ⊢$_B$ *q*))›
unfolding *consistent-def* **using** *K-Boole K-imply-Cons* **by** *blast*
next
fix *A B* **and** *p q* :: ‹*'p fm*›
assume ‹*A* ⊢$_B$ *p*› ‹*p* # *B* ⊢$_B$ *q*›
then show ‹*A* @ *B* ⊢$_B$ *q*›
by (*metis K-right-mp add-imply imply.simps*(*2*) *imply-append*)
qed *simp*

interpretation *Derivations-Bot consistent Calculus-assms* ‹⊥›
proof
show ‹⋀*A r*. *A* ⊢$_B$ ⊥ ⟹ *A* ⊢$_B$ *r*›
using *K-Boole K-imply-Cons* **by** *blast*
qed

interpretation *Derivations-Imp consistent Calculus-assms* ‹λ*p q*. *p* ⟶ *q*›
proof
show ‹⋀*A p q*. *p* # *A* ⊢$_B$ *q* ⟹ *A* ⊢$_B$ *p* ⟶ *q*›
using *K-ImpI* **by** *blast*
show ‹⋀*A p q*. *A* ⊢$_B$ *p* ⟹ *A* ⊢$_B$ *p* ⟶ *q* ⟹ *A* ⊢$_B$ *q*›
using *K-right-mp* **by** *blast*
qed

theorem *deriv-in-maximal*:
assumes ‹*consistent S*› ‹*maximal S*› ‹⊢$_B$ *p*›
shows ‹*p* ∈ *S*›
using *assms MCS-derive* **by** *fastforce*

lemma *dia-not-box-bot*:
assumes ‹*consistent S*› ‹*maximal S*› ‹$\langle b \rangle\ F \in S$›
shows ‹$\neg[b]\bot \in S$›
proof −
have ‹$\neg[b]\neg F \in S$› **using** *assms*(*3*) **by** *simp*
have ‹$\bot \longrightarrow \neg F \in S$› **by** (*simp add*: *MCS-impI assms*(*1*,*2*))
hence ‹$[b](\bot \longrightarrow \neg F) \in S$› **using** *Calculus.Nb*
by (*metis K-imply-head assms*(*1*,*2*) *deriv-in-maximal imply.simps*(*1*,*2*))
hence ‹$[b]\bot \longrightarrow [b]\neg F \in S$›
by (*meson Kb MCS-imp assms*(*1*,*2*) *deriv-in-maximal*)
thus ‹$\neg[b]\bot \in S$› **using** ‹$\neg[b]\neg F \in S$›
by (*simp add*: *MCS-imp assms*(*1*,*2*))
qed

Some other useful lemmas that are repeatedly used in proofs.

lemma *not-empty*:
assumes ‹$a \in A$›
shows ‹$A \neq \{\}$›
using *assms* **by** *auto*

lemma *MPcons*:
assumes ‹$\vdash_B A \longrightarrow (B \longrightarrow C)$›
and ‹$\vdash_B B$›
shows ‹$\vdash_B A \longrightarrow C$›
by (*metis K-right-mp add-imply assms*(*1*,*2*) *imply.simps*(*1*,*2*))

lemma *multiple-MP-MCS*:
assumes ‹*MCS S*›
and ‹*set A* $\subseteq$ *S*›
and ‹$A \rightsquigarrow f \in S$›
shows ‹$f \in S$› **using** *assms* **by** (*induct A*) *auto*

lemma *not-imp-to-conj*:
assumes ‹*MCS A*›
and ‹$\neg(B \rightsquigarrow \bot) \in A$›
shows ‹*set B* $\subseteq$ *A*›
proof (*rule ccontr*)
assume ‹$\neg$ *set B* $\subseteq$ *A*›
then obtain *f* **where** *f*: ‹$f \in$ *set B* $\wedge$ $f \notin$ *A*› **by** *auto*
hence ‹$(B \rightsquigarrow \bot) \in A$›
by (*smt* (*verit*, *ccfv-SIG*) *MCS-derive MCS-explode assms*(*1*) *derive-assm*
derive-cut imply-append imply-swap-append)
thus *False* **using** *assms*(*2*)
using *MCS-bot assms*(*1*) **by** *blast*
qed

Several lemmas of *Kop*, valid for normal modal operators.

context *Kop* **begin**

lemma *not-pos-to-nec-not*:
shows ‹$\vdash_B \neg\mathbf{P}F \longrightarrow \mathbf{K}\neg F$›
proof −
have *P1*: ‹$\vdash_B ((\mathbf{P}F \longleftrightarrow \neg\mathbf{K}\neg F) \longrightarrow (\neg\mathbf{P}F \longrightarrow \mathbf{K}\neg F))$› **using** *TAU* **by** *force*
have ‹$\vdash_B \mathbf{P}F \longleftrightarrow \neg\mathbf{K}\neg F$› **using** *TAU* **by** *force*
thus *?thesis* **using** *P1 MP* **by** *auto*
qed

lemma *not-pos-to-nec-not-deriv*:
assumes ‹$\vdash_B \neg F \longrightarrow G$›
shows ‹$\vdash_B \neg\mathbf{P}F \longrightarrow \mathbf{K}G$›
by (*metis KN K-imply-Cons K-right-mp Kax assms*
imply.simps(*1*,*2*) *not-pos-to-nec-not*)

lemma *pos-not-to-not-nec*:
shows ‹$\vdash_B \mathbf{P}\neg F \longrightarrow \neg\mathbf{K}F$›
proof −
have *P1*: ‹$\vdash_B \mathbf{P}\neg F \longleftrightarrow \neg\mathbf{K}\neg\neg F$› **using** *TAU* **by** *force*
have ‹$\vdash_B \mathbf{K}(F \longrightarrow \neg\neg F)$› **using** *TAU KN* **by** *force*
hence *P2*: ‹$\vdash_B \mathbf{K}F \longrightarrow \mathbf{K}\neg\neg F$› **using** *Kax MP* **by** *auto*
have ‹$\vdash_B (\mathbf{P}\neg F \longleftrightarrow \neg\mathbf{K}\neg\neg F) \longrightarrow (\mathbf{K}F \longrightarrow \mathbf{K}\neg\neg F) \longrightarrow (\mathbf{P}\neg F \longrightarrow \neg\mathbf{K}F)$›

using *TAU* **by** *force*
thus *?thesis* **using** *P1 P2 MP* **by** *auto*
qed

lemma *not-nec-to-pos-not*:
shows ‹$\vdash_B \neg\mathbf{K}F \longrightarrow \mathbf{P}\neg F$›
proof −
have *P1*: ‹$\vdash_B \mathbf{P}\neg F \longleftrightarrow \neg\mathbf{K}\neg\neg F$› **using** *TAU* **by** *force*
have ‹$\vdash_B \mathbf{K}(\neg\neg F \longrightarrow F)$› **using** *TAU KN* **by** *force*
hence *P2*: ‹$\vdash_B \mathbf{K}\neg\neg F \longrightarrow \mathbf{K}F$› **using** *Kax MP* **by** *auto*
have ‹$\vdash_B (\mathbf{P}\neg F \longleftrightarrow \neg\mathbf{K}\neg\neg F) \longrightarrow (\mathbf{K}\neg\neg F \longrightarrow \mathbf{K}F) \longrightarrow (\neg\mathbf{K}F \longrightarrow \mathbf{P}\neg F)$›

using *TAU* **by** *force*
thus *?thesis* **using** *P1 P2 MP* **by** *auto*
qed

lemma *pos-not-to-not-nec-MCS*:
assumes ‹*MCS A*›
and ‹$\mathbf{P}\neg F \in A$›
shows ‹$\neg\mathbf{K}F \in A$› **using** *pos-not-to-not-nec*
by (*meson MCS-imp assms*(*1*,*2*) *deriv-in-maximal*)

lemma *pos-subset*:
assumes ‹*MCS A*› **and** ‹*MCS B*›
shows ‹$\{ F \mid F . \mathbf{K}\ F \in A\} \subseteq B \longleftrightarrow \{\mathbf{P}F \mid F . F \in B\} \subseteq A$›
proof
assume *As*: ‹$\{ F \mid F . \mathbf{K}\ F \in A\} \subseteq B$›

```
  show ‹{P F | F . F ∈ B} ⊆ A›
  proof
    fix F
    assume ‹F ∈ {P F |F. F ∈ B}›
    then obtain G where P: ‹F = P G ∧ G∈B› by auto
    show ‹F ∈ A›
    proof (rule ccontr)
      assume ‹F ∉ A›
      hence ‹K¬G ∈ A› using assms(1) P ‹F ∉ A›
        by (metis (no-types, opaque-lifting) MCS-impI)
      thus False using As P assms(2) MCS-bot by blast
    qed
  qed
next
  assume As: ‹{P F | F . F ∈ B} ⊆ A›
  show ‹{ F | F . K F ∈ A} ⊆ B›
  proof
    fix F
    assume ‹F ∈ { F | F . K F ∈ A}›
    hence ‹K F ∈ A› by simp
    show ‹F ∈ B›
    proof (rule ccontr)
      assume ‹F ∉ B›
      hence ‹¬KF ∈  A›
        using As assms pos-not-to-not-nec-MCS by auto
      thus False
        using ‹K F ∈ A› MCS-bot assms(1) by blast
    qed
  qed
qed

end
```

Lemmas involving the negation of a chain of $\Box$ or $\Diamond$.

```
lemma not-chain-d-to-chain-b-not:
  assumes ‹⊢B ¬F ⟶ G›
  shows ‹⊢B ¬ (◇^n F) ⟶ (□^n G)›
proof (induct n)
  case 0
  then show ?case using assms by auto
next
  case (Suc n)
  assume ‹⊢B ¬ (◇^n F) ⟶ (□^n G)›
  hence H: ‹⊢B □¬(◇^n F) ⟶ □(□^n G)› using KSq MP NSq by blast
  have H2: ‹⋀ A B C . ⊢B (A ⟶ B) ⟶ (B ⟶ C) ⟶ (A ⟶ C)› using
TAU by force
  have ‹⊢B ¬(◇◇^n F) ⟶ □¬(◇^n F)›
    using KBox.not-pos-to-nec-not chain-d-def by auto
  hence ‹⊢B ¬(◇◇^n F) ⟶ □(□^n G)› using H H2 MP by blast
```

thus *?case* **using** *chain-b-def chain-d-def* **by** *simp*
qed

lemma *not-chain-b-to-chain-d-not*:
assumes ‹$\vdash_B \neg F \longrightarrow G$›
shows ‹$\vdash_B \neg\ (\Box\hat{}n\ F) \longrightarrow (\Diamond\hat{}n\ G)$›
proof (*induct n*)
case *0*
then show *?case* **using** *assms* **by** *auto*
next
case (*Suc n*)
assume *P1*: ‹$\vdash_B \neg\ (\Box\hat{}n\ F) \longrightarrow (\Diamond\hat{}n\ G)$›
have *T*: ‹$\bigwedge A\ B\ .\ \vdash_B (A \longrightarrow B)\ \longrightarrow (\neg\ B \longrightarrow \neg\ A)$› **using** *TAU* **by** *force*
hence ‹$\vdash_B \neg\ (\Diamond\hat{}n\ G) \longrightarrow \neg\neg\ (\Box\hat{}n\ F)$› **using** *P1 MP* **by** *auto*
hence ‹$\vdash_B \Box\neg(\Diamond\hat{}n\ G) \longrightarrow \Box\neg\neg(\Box\hat{}n\ F)$› **using** *KSq MP NSq* **by** *blast*
hence *P2*: ‹$\vdash_B \Diamond\neg(\Box\hat{}n\ F) \longrightarrow \Diamond(\Diamond\hat{}n\ G)$› **using** *T MP* **by** *auto*
have *P3*: ‹$\vdash_B \neg\ (\Box\hat{}(n+1)\ F) \longrightarrow \Diamond\neg(\Box\hat{}n\ F)$›
using *KBox.not-nec-to-pos-not* **by** *auto*
have ‹$\bigwedge A\ B\ C\ .\ \vdash_B (A \longrightarrow B) \longrightarrow (B \longrightarrow C) \longrightarrow (A \longrightarrow C)$› **using** *TAU*
by *force*
hence ‹$\vdash_B (\neg\Box\hat{}(n+1)\ F) \longrightarrow \Diamond(\Diamond\hat{}n\ G)$› **using** *P2 P3 MP* **by** *blast*
thus *?case* **using** *chain-b-def P2 MP chain-d-def* **by** *simp*
qed

lemma *not-chain-b-to-chain-d-not-rev*:
assumes ‹$\vdash_B F \longrightarrow \neg G$›
shows ‹$\vdash_B (\Diamond\hat{}n\ G) \longrightarrow\ \neg\ (\Box\hat{}n\ F)$›
proof (*induct n*)
case *0*
have ‹$\vdash_B (F \longrightarrow \neg G) \longrightarrow (G \longrightarrow \neg F)$› **using** *TAU* **by** *force*
then show *?case* **using** *assms chain-b-def chain-d-def MP* **by** *auto*
next
case (*Suc n*)
assume *P1*: ‹$\vdash_B (\Diamond\hat{}n\ G) \longrightarrow\ \neg\ (\Box\hat{}n\ F)$›
have *T*: ‹$\bigwedge A\ B\ .\ \vdash_B (A \longrightarrow \neg B)\ \longrightarrow (B \longrightarrow \neg\ A)$› **using** *TAU* **by** *force*
hence ‹$\vdash_B (\Box\hat{}n\ F) \longrightarrow \neg(\Diamond\hat{}n\ G)$› **using** *P1 MP* **by** *auto*
hence ‹$\vdash_B \Box(\Box\hat{}n\ F) \longrightarrow \Box\neg(\Diamond\hat{}n\ G)$› **using** *KSq MP NSq* **by** *blast*
hence *P2*: ‹$\vdash_B (\Box\hat{}(n+1)\ F) \longrightarrow \Box\neg(\Diamond\hat{}n\ G)$› **using** *T MP chain-b-def* **by**
simp
have ‹$\bigwedge A\ B\ .\ \vdash_B (A \longrightarrow B)\ \longrightarrow (\neg B \longrightarrow \neg\ A)$› **using** *TAU* **by** *force*
hence *P3*: ‹$\vdash_B \neg\Box\neg(\Diamond\hat{}n\ G) \longrightarrow \neg(\Box\hat{}(n+1)\ F)$› **using** *P2 MP* **by** *blast*
thus *?case* **using** *chain-d-def* **by** *simp*
qed

7 Elements for the Canonical model

First, we introduce some relations that will by used to define the components of the Canonical Model. The first one is the chain relation *Chn*:

abbreviation *Chn* :: ‹('*p fm set* × '*p fm set*) *set*› **where**
‹*Chn* ≡ {(*Sa*,*Sb*) . *MCS Sa* ∧ *MCS Sb* ∧ {*f* . □*f* ∈ *Sa*} ⊆ *Sb*}›

Now, the eelation *Zmc* linking MCS that will produce the bisimilarity relation:

abbreviation *Zmc* :: ‹('*p fm set* × '*p fm set*) *set*› **where**
‹*Zmc* ≡ {(*Sa*,*Sb*) . *MCS Sa* ∧ *MCS Sb* ∧ {*f*. [*b*]*f* ∈ *Sa*} ⊆ *Sb*}›

Truth of propositional variables in MCS:

abbreviation *Vmc* :: ‹'*p fm set* ⇒ '*p* ⇒ *bool*› **where**
‹*Vmc* ≡ (λ *S P*. •*P* ∈ *S*)›

Sets *MC1* and *MC2* will constitute the worlds on the left and on the right model of the canonical model for $\mathcal{L}_{\Box[b]}$. All mc-sets are in *MC1*, while *MC2* contains only mc-sets containing $\Box\hat{}n[b]\bot$ for all *n*.

abbreviation *MC1* :: ‹'*p fm set set*› **where**
‹*MC1* ≡ {*A* . *MCS A*}›

abbreviation *MC2* :: ‹'*p fm set set*› **where**
‹*MC2* ≡ {*A* . *MCS A* ∧ (∀ *n* . □^*n* [*b*]⊥ ∈ *A*)}›

This lemma shows that *Zmc* goes from worlds not in *MC2* to worlds in *MC2*.

lemma *Z-from-MC1-to-MC2*:
assumes ‹(*A*,*B*) ∈ *Zmc*›
shows ‹*A* ∉ *MC2* ∧ *B* ∈ *MC2*›
proof −
have ‹⊤ ∈ *B*› **using** *assms* **by** *auto*
hence ‹⟨*b*⟩⊤ ∈ *A*› **using** *KFrB.pos-subset assms* **by** *force*
have ‹*A* ∉ *MC2*›
proof
assume ‹*A* ∈ *MC2*›
hence ‹[*b*]⊥ ∈ *A*› **by** (*metis* (*no-types*, *lifting*) *mem-Collect-eq chain-b.simps*(*1*))
thus *False* **using** ‹⟨*b*⟩⊤ ∈ *A*› *assms* **by** *fastforce*
qed
have ‹∀ *n* . [*b*] □^*n* [*b*] ⊥ ∈ *A*› **using** *assms NTSgen deriv-in-maximal* **by** *auto*
hence ‹∀ *n* . □^*n* [*b*] ⊥ ∈ *B*› **using** *assms* **by** *auto*
hence ‹*B* ∈ *MC2*› **using** *assms* **by** *blast*
thus *?thesis* **using** ‹*A* ∉ *MC2*› **by** *simp*
qed

The following lemma is important for the proof of existence. It proves that if {*f* . □*f* ∈ *A*} ⊆ *B* and *A* ∈ *MC2*, then *B* ∈ *MC2*.

lemma *Chn-from-to-2*:

assumes ‹$(A,B) \in Chn$› **and** ‹$A \in MC2$›
shows ‹$B \in MC2$›
proof −
have *Amc2*: ‹$\forall\ n\ .\ \Box\hat{}n\ [b]\bot \in A$›
using *assms(2)* **by** *auto*
have ‹$\forall\ m\ .\ \Box\hat{}(m+1)\ [b]\ \bot \in A \longrightarrow \Box\hat{}m\ [b]\ \bot \in B$› **using** *assms(1)* **by** *force*
hence ‹$\forall\ n\ .\ \Box\hat{}n\ [b]\bot \in B$› **using** *Amc2* **by** *blast*
thus *?thesis* **using** *assms* **by** *blast*
qed

Lemmas used to prove existence.

lemma *pos-r1-sub*:
assumes ‹$A \in MC1$› **and** ‹$B \in MC1$›
shows ‹$(A,B) \in Chn \longleftrightarrow \{\Diamond F \mid F\ .\ F \in B\} \subseteq A$›
proof −
have ‹$MCS\ A$› **using** *assms(1)* **by** *simp*
have ‹$MCS\ B$› **using** *assms(2)* **by** *simp*
have *P*: ‹$(A,B) \in Chn \longleftrightarrow \{f\ .\ \Box f \in A\} \subseteq B$› **using** *assms* **by** *simp*
have ‹$(\{F\ |F.\ \Box\ F \in A\} \subseteq B) \longleftrightarrow (\{\Diamond\ F\ |F.\ F \in B\} \subseteq A)$›
using ‹$MCS\ A$› ‹$MCS\ B$› *KBox.pos-subset* **by** *simp*
thus *?thesis* **using** *P* **by** *simp*
qed

lemma *pos-r2-sub*:
assumes ‹$A \in MC2$› **and** ‹$B \in MC2$›
shows ‹$(A,B) \in Chn \longleftrightarrow \{\Diamond F \mid F\ .\ F \in B\} \subseteq A$›
proof −
have ‹$MCS\ A$› **using** *assms(1)* **by** *simp*
have ‹$MCS\ B$› **using** *assms(2)* **by** *simp*
have *P*: ‹$(A,B) \in Chn \longleftrightarrow \{f\ .\ \Box f \in A\} \subseteq B$› **using** *assms* **by** *simp*
have ‹$(\{F\ |F.\ \Box\ F \in A\} \subseteq B) \longleftrightarrow (\{\Diamond\ F\ |F.\ F \in B\} \subseteq A)$›
using ‹$MCS\ A$› ‹$MCS\ B$› *KBox.pos-subset* **by** *simp*
thus *?thesis* **using** *P* **by** *simp*
qed

lemma *pos-b-r2-sub*:
assumes ‹$A \in MC1$› **and** ‹$B \in MC2$›
shows ‹$(A,B) \in Zmc \longleftrightarrow \{\langle b\rangle F \mid F\ .\ F \in B\} \subseteq A$›
proof −
have ‹$MCS\ A$› **using** *assms(1)* **by** *simp*
have ‹$MCS\ B$› **using** *assms(2)* **by** *simp*
have *P*: ‹$(A,B) \in Zmc \longleftrightarrow \{f\ .\ [b]f \in A\} \subseteq B$› **using** *assms* **by** *simp*
have ‹$(\{F\ |F.\ [b]F \in A\} \subseteq B) \longleftrightarrow (\{\ \langle b\rangle\ F\ |F.\ F \in B\} \subseteq A)$›
using ‹$MCS\ A$› ‹$MCS\ B$› *KFrB.pos-subset* **by** *simp*
thus *?thesis* **using** *P* **by** *simp*
qed

All mc-sets in *MC2* contain $[b]F$ for every F.

lemma *all-box-b-in-MC2*:

assumes ‹$S \in MC2$›
shows ‹$[b]F \in S$›
proof −
have ‹$\vdash_B \bot \longrightarrow F$› **using** *TAU* **by** *force*
have ‹$[b]\bot \in S$› **using** *assms chain-b.simps(1)*
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq*)
thus *?thesis* **using** ‹$\vdash_B \bot \longrightarrow F$›
by (*smt* (*verit*, *best*) *KFrB.KN KFrB.Kax MCS-imp assms deriv-in-maximal mem-Collect-eq*)
qed

8 Existence

First, we prove a general result for all normal modal operators.

context *Kop* **begin**

lemma *Kop-existence*:
assumes ‹*MCS A*›
and ‹$\mathbf{P}F \in A$›
shows ‹*consistent* ($\{F\} \cup \{G$. $\mathbf{K}G \in A\}$)›
proof (*rule ccontr*)
assume ‹$\neg$ *consistent* ($\{F\} \cup \{G$. $\mathbf{K}G \in A\}$)›
then obtain S **where** S: ‹*set* $S \subseteq \{G$. $\mathbf{K}G \in A\} \wedge F\#S \vdash_B \bot$›
by (*metis* (*no-types*, *lifting*) *K-imply-Cons consistent-def derive-split1 insert-is-Un subset-insert*)
hence ‹$S \vdash_B \neg F$›
using *K-ImpI* **by** *blast*
hence ‹$\vdash_B S \rightsquigarrow \neg F$› **by** *simp*
hence D: ‹$\vdash_B$ *map* (λf . $\mathbf{K}f$) $S \rightsquigarrow \mathbf{K}\neg F$›
using *KN K-distrib-K-imp* **by** *blast*
have ‹*set*(*map* (λf . $\mathbf{K}f$) S) $\subseteq A$› **using** S **by** *auto*
hence ‹$\mathbf{K}\neg F \in A$› **using** D
by (*meson MCS-derive assms(1)*)
hence ‹$\neg\mathbf{K}\neg F \notin A$›
using *MCS-bot assms(1)* **by** *blast*
thus *False* **using** *assms(2)* **by** *auto*
qed

end

lemma *Chn-iff*:
assumes ‹*MCS A*›
shows ‹$\Box F \in A \longleftrightarrow (\forall\ B$. $(A,B) \in Chn \longrightarrow F \in B)$›
proof
assume ‹$\Box F \in A$›
thus ‹$\forall\ B$. $(A,B) \in Chn \longrightarrow F \in B$› **by** *auto*
next

assume *A*: ‹∀ *B* . (*A*,*B*) ∈ *Chn* ⟶ *F*∈*B*›
show ‹□*F* ∈ *A*›
proof (*rule ccontr*)
assume ‹□*F* ∉ *A*›
hence ‹◇ ¬*F* ∈ *A*›
by (*meson KBox.not-nec-to-pos-not MCS-imp assms deriv-in-maximal*)
hence ‹*consistent* ({¬*F*} ∪ {*G* . □ *G* ∈ *A*})›
using ‹*MCS A*› *KBox.Kop-existence* **by** *auto*
hence ‹∃ *S* . (*MCS S* ∧ ({¬*F*} ∪ {*G* . □ *G* ∈ *A*}) ⊆ *S*)›
using *Extend-subset MCS-Extend′* **by** *metis*
then obtain *S* **where** *S*: ‹(*A*,*S*) ∈ *Chn* ∧ ¬*F* ∈ *S* ∧ *F* ∈ *S*› **using** *A assms*
by *auto*
thus *False* **using** *MCS-bot* **by** *blast*
qed
qed

Existence for ◇ in *MC1*.

lemma *existenceChn-1*:
assumes ‹◇*F* ∈ *A*› **and** ‹*A* ∈ *MC1*›
shows ‹∃ *B* . *B* ∈ *MC1* ∧ {*F*} ∪ {*G* . □*G* ∈ *A*}⊆*B*›
proof −
have ‹*consistent* ({*F*} ∪ {*G* . □*G* ∈ *A*})› **using** *assms KBox.Kop-existence* **by**
auto
then obtain *S* **where** *S*: ‹*MCS S* ∧ ({*F*} ∪ {*G* . □*G* ∈ *A*}) ⊆ *S*›
by (*metis Extend-subset MCS-Extend′*)
hence ‹(*A*,*S*) ∈ *Chn*› **using** *assms* **by** *simp*
thus *?thesis* **using** *S* **by** *auto*
qed

Existence for ◇ in *MC2*.

lemma *existenceChn-2*:
assumes ‹◇*F* ∈ *A*› **and** ‹*A* ∈ *MC2*›
shows ‹∃ *B* . *B* ∈ *MC2* ∧ {*F*} ∪ {*G* . □*G* ∈ *A*}⊆*B*›
proof −
have ‹*consistent* ({*F*} ∪ {*G* . □*G* ∈ *A*})› **using** *assms KBox.Kop-existence* **by**
auto
then obtain *S* **where** *S*: ‹*MCS S* ∧ ({*F*} ∪ {*G* . □*G* ∈ *A*}) ⊆ *S*›
by (*metis Extend-subset MCS-Extend′*)
hence ‹(*A*,*S*) ∈ *Chn*› **using** *assms* **by** *simp*
thus *?thesis*
by (*metis* (*mono-tags*, *lifting*) *Chn-from-to-2 S assms*(*2*))
qed

Existence for ⟨*b*⟩ in *MC1*.

lemma *existenceZmc*:
assumes ‹⟨*b*⟩*F* ∈ *A*› **and** ‹*A* ∈ *MC1*›
shows ‹∃ *B* . *B* ∈ *MC2* ∧ {*F*} ∪ {*G* . [*b*]*G* ∈ *A*}⊆*B*›
proof −

have ‹*consistent* ($\{F\} \cup \{G$. $[b]G \in A\}$)› **using** *assms KFrB.Kop-existence* **by** *auto*
then obtain *S* **where** *S*: ‹*MCS S* $\wedge$ ($\{F\} \cup \{G$. $[b]G \in A\}$) $\subseteq$ *S*›
by (*metis Extend-subset MCS-Extend'*)
hence ‹(*A*,*S*) $\in$ *Zmc*› **using** *assms* **by** *simp*
thus *?thesis*
by (*metis* (*mono-tags*, *lifting*) *S Z-from-MC1-to-MC2*)
qed

9 Atomic harmony of *Zmc*.

MCS linked by *Zmc* contain the same propositional variables.

lemma *Zmc-atomic-harmony*:
assumes ‹(*A*,*B*) $\in$ *Zmc*›
shows ‹$\cdot l \in A \longleftrightarrow \cdot l \in B$›
proof
assume ‹$\cdot l \in A$›
hence ‹$[b]\cdot l \in A$›
by (*metis* (*mono-tags*, *lifting*) *HARM assms deriv-in-maximal Derivations-Imp.MCS-impE*
Derivations-Imp-axioms mem-Collect-eq old.prod.case)
thus ‹$\cdot l \in B$› **using** *assms* **by** *auto*
next
assume ‹$\cdot l \in B$›
thus ‹$\cdot l \in A$›
by (*smt* (*verit*, *del-insts*) *HARM MCS-bot MCS-imp assms deriv-in-maximal*
mem-Collect-eq prod.simps(*2*) *subsetD*)
qed

10 Forth and Back

First, an auxiliary lemma used in the proofs of forth and back properties of the Canonical Model.

lemma *nec-As-to-nec-conj*:
assumes ‹*S* $\in$ *MC1*›
and ‹$\{[b]f \mid f.\ f \in$ *set A*$\} \subseteq$ *S*›
shows ‹$[b]\neg(A \rightsquigarrow \bot) \in S$›
proof (*rule ccontr*)
assume ‹$[b]\ \neg(A \rightsquigarrow \bot) \notin S$›
hence ‹$\neg[b]\ \neg\ (A \rightsquigarrow \bot) \in S$› **using** *assms*(*1*) **by** *blast*
hence ‹$\langle b\rangle\ (A \rightsquigarrow \bot) \in S$› **by** *simp*
then obtain *S2* **where** *S2*: ‹(*MCS S2*) $\wedge$ ($\{A \rightsquigarrow \bot\} \cup \{G$. $[b]G \in S\}$) $\subseteq$ *S2*›
using *assms*(*1*)
existenceZmc **by** (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)
hence ‹($\{A \rightsquigarrow \bot\} \cup$ *set A*) $\subseteq$ *S2*› **using** *assms*(*2*) *mem-Collect-eq* **by** *auto*
hence ‹$\bot \in$ *S2*› **using** *multiple-MP-MCS S2* **by** (*metis insert-is-Un insert-subset*)
thus *False* **using** *multiple-MP-MCS S2 insert-is-Un insert-subset* **by** *simp*

qed

Lemmas that will be used to prove that the Canonical Model satisfies forth and back properties.

lemma *forth-cm*:
assumes ‹(*G1*, *G2*) ∈ *Zmc*›
and ‹(*G1*, *G3*) ∈ *Chn*›
shows ‹∃ *G4* . (*G3*, *G4*) ∈ *Zmc* ∧ (*G2*, *G4*) ∈ *Chn*›
proof −
have *GS*: ‹*G1* ∈ *MC1* ∧ *G2* ∈ *MC2* ∧ *G3* ∈ *MC1*›
using *assms Z-from-MC1-to-MC2* **by** *auto*
have ‹*consistent* ({*A* . [*b*]*A* ∈ *G3*} ∪ {*B* | *B*. □*B* ∈ *G2*})› (**is** ‹*consistent* (*?L* ∪ *?R*)›)
proof (*rule ccontr*)
assume ‹¬ *consistent* (*?L* ∪ *?R*)›
then obtain *AsBs* **where** *AB1*: ‹*set AsBs* ⊆ *?L* ∪ *?R* ∧ (*AsBs* ⊢$_B$ ⊥)›
using *consistent-def* **by** *blast*
then obtain *As Bs* **where** *AB*: ‹*set As* = (*set AsBs* ∩ *?L*) ∧
set Bs = (*set AsBs* ∩ *?R*)›
by (*metis* (*lifting*) *inf-commute inter-set-filter*)
hence ‹*set As* ∪ *set Bs* = *set AsBs*› **using** *AB1* **by** *auto*
hence ‹(*Bs*@*As* ⊢$_B$ ⊥)› **using** *AB*
by (*metis AB1 K-imply-weaken dual-order.refl set-append sup.commute*)
hence ‹⊢$_B$ *Bs* ⇝ *As* ⇝ ⊥› **by** *simp*
hence *D*: ‹⊢$_B$ (*map* (λ *f*. □ *f*) *Bs*) ⇝ □(*As* ⇝ ⊥)›
using *KBox.KN KBox.K-distrib-K-imp* **by** *blast*
have ‹*finite* (*set As*)› **using** ‹(*Bs*@*As* ⊢$_B$ ⊥)› **by** *simp*
have ‹*set* (*map* (λ *f*. □ *f*) *Bs*) ⊆ *G2*› **using** *AB* **by** *auto*
hence ‹□(*As* ⇝ ⊥) ∈ *G2*› **using** *D*
by (*metis* (*no-types*, *lifting*) *MCS-derive assms*(*1*) *case-prod-conv*
mem-Collect-eq)
hence *f1*: ‹⟨*b*⟩□(*As* ⇝ ⊥) ∈ *G1*› (**is** ‹⟨*b*⟩□*?F1* ∈ *G1*›)
by (*metis* (*mono-tags*, *lifting*) *MCS-bot MCS-impE MCS-impI assms*(*1*)
mem-Collect-eq prod.simps(*2*) *subsetD*)
obtain *NecAs* **where** *NecAs*: ‹*NecAs* = (*map* (λ *f*. [*b*]*f*) *As*)› **by** *simp*
hence *sAs*: ‹*set NecAs* ⊆ *G3* ∧ *finite* (*set NecAs*)›
using *AB assms* **by** *auto*
hence ‹{[*b*] *f* |*f*. *f* ∈ *set As*} ⊆ *G3*› **using** *AB* **by** *blast*
hence ‹[*b*]¬(*As* ⇝ ⊥) ∈ *G3*› **using** *GS nec-As-to-nec-conj*[*of G3 As*] **by** *simp*
hence ‹◇[*b*]¬ (*As* ⇝ ⊥) ∈ *G1*› (**is** ‹◇[*b*] *?F2* ∈ *G1*›) **using** *assms KBox.pos-subset*
by *force*
hence ‹⟨*b*⟩□*?F1* ∧ ◇[*b*] *?F2* ∈ *G1*›
using *f1 Derivations-Imp.MCS-imp Derivations-Imp-axioms assms*(*1*) **by**
fastforce
hence ‹⟨*b*⟩(□*?F1* ∧ ◇ *?F2*) ∈ *G1*› (**is** ‹⟨*b*⟩ *?F12* ∈ *G1*›) **using** *FORTH MP*
Derivations-Imp.MCS-imp Derivations-Imp-axioms GS deriv-in-maximal **by**
fastforce
then obtain *A* **where** *A*: ‹(*G1*,*A*)∈*Zmc* ∧ *A* ∈ *MC2* ∧ (□*?F1* ∧ ◇*?F2* ∈ *A*)›

using *assms*(1) *existenceZmc*[*of ?F12 G1*] *Z-from-MC1-to-MC2*[*of G1*]
by *auto*
hence ‹{□*?F1*, ◇*?F2*} ⊆ *A*›
by (*metis* (*mono-tags*, *lifting*) *MCS-bot MCS-imp empty-subsetI insert-subset mem-Collect-eq*)
then obtain *B* **where** *B*: ‹*B* ∈ *MC2* ∧ {*?F1*, *?F2*} ⊆ *B*›
using *A existenceChn-2*[*of ?F2*]
by (*metis* (*mono-tags*, *lifting*) *KBox.pos-not-to-not-nec-MCS MCS-bot MCS-imp mem-Collect-eq*)
hence ‹⊥ ∈ *B*› **by** *blast*
thus *False* **using** *B* **by** *auto*
qed
then obtain *G4* **where** *G4*: ‹(*?L* ∪ *?R*) ⊆ *G4* ∧ *MCS G4*›
using *Extend-subset MCS-Extend′* **by** *fastforce*
hence ‹{*B* | *B*. □*B* ∈ *G2*} ⊆ *G4*› **by** *simp*
hence *T1*: ‹(*G2*, *G4*) ∈ *Chn*› **using** *G4 GS* **by** *simp*
hence ‹(*G3*, *G4*) ∈ *Zmc*› **using** *G4 GS Chn-from-to-2* **by** *auto*
thus *?thesis* **using** *T1* **by** *auto*
qed

lemma *back-cm*:
assumes ‹(*G1*, *G2*) ∈ *Zmc*›
and ‹(*G2*, *G3*) ∈ *Chn*›
shows ‹∃ *G4* . (*G1*, *G4*) ∈ *Chn* ∧ (*G4*, *G3*) ∈ *Zmc*›
proof −
have *GS*: ‹*G1* ∈ *MC1* ∧ *G2* ∈ *MC2* ∧ *G3* ∈ *MC2*›
using *assms Chn-from-to-2 Z-from-MC1-to-MC2*
by (*metis* (*mono-tags*, *lifting*) *case-prodD mem-Collect-eq*)
have ‹*consistent* ({⟨*b*⟩*A* | *A*. *A* ∈ *G3*} ∪ {*B* | *B*. □*B* ∈ *G1*})› (**is** ‹*consistent* (*?L* ∪ *?R*)›)
proof (*rule ccontr*)
assume ‹¬ *consistent* (*?L* ∪ *?R*)›
then obtain *PosAsBs* **where** *AB1*: ‹*set PosAsBs* ⊆ *?L* ∪ *?R* ∧ (*PosAsBs* ⊢$_B$ ⊥)›
using *consistent-def* **by** *blast*
then obtain *PosAs Bs* **where** *AB*: ‹*set PosAs* = (*set PosAsBs* ∩ *?L*) ∧ *set Bs* = (*set PosAsBs* ∩ *?R*)›
by (*metis* (*lifting*) *inf-commute inter-set-filter*)
hence ‹*set PosAs* ∪ *set Bs* = *set PosAsBs*› **using** *AB1* **by** *auto*
hence ‹(*Bs*@*PosAs* ⊢$_B$ ⊥)› **using** *AB*
by (*metis AB1 K-imply-weaken dual-order.refl set-append sup.commute*)
hence ‹⊢$_B$ *Bs* ⇝ *PosAs* ⇝ ⊥› **by** *simp*
hence *D*: ‹⊢$_B$ (*map* (λ *f*. □ *f*) *Bs*) ⇝ □(*PosAs* ⇝ ⊥)›
using *KBox.KN KBox.K-distrib-K-imp* **by** *blast*
have ‹*finite* (*set PosAs*)› **using** ‹(*Bs*@*PosAs* ⊢$_B$ ⊥)› **by** *simp*
have ‹*set* (*map* (λ *f*. □ *f*) *Bs*) ⊆ *G1*› **using** *AB* **by** *auto*
hence *X1*: ‹□(*PosAs* ⇝ ⊥) ∈ *G1*› **using** *D*
by (*metis* (*no-types*, *lifting*) *MCS-derive assms*(1) *case-prod-conv*

mem-Collect-eq)
obtain *As* **where** *As*: ‹*As* = (*map* (λ *f*. *THE p* . *f* = ⟨*b*⟩*p*) *PosAs*)› **by** *simp*
hence *sAs*: ‹*set As* ⊆ *G3* ∧ *finite* (*set As*)›
using *AB assms* **by** *auto*
have ‹∀ *f* . *f* ∈ *set PosAs* ⟶ (∃ *g* . *f* = ⟨*b*⟩*g*)› **using** *AB* **by** *auto*
hence *X*: ‹*set PosAs* = {⟨*b*⟩*g* | *g*. *g* ∈ *set As*}› **using** *As* **by** *force*
have ‹*As* ⇝ ⊥ ∉ *G3*› **using** *multiple-MP-MCS MCS-bot assms*(*2*) *sAs* **by** *auto*
hence ‹¬ (*As* ⇝ ⊥) ∈ *G3*› (**is** ‹*?F2* ∈ *G3*›) **using** *assms*(*2*) **by** *blast*
hence ‹◇*?F2* ∈ *G2*›
by (*smt* (*verit*, *ccfv-threshold*) *assms KBox.pos-subset GS*
mem-Collect-eq pos-r2-sub subsetD)
hence ‹⟨*b*⟩◇*?F2* ∈ *G1*› **using** *assms KFrB.pos-subset* **by** *fastforce*
hence *X2*: ‹◇⟨*b*⟩ *?F2* ∈ *G1*› **by** (*metis* (*mono-tags*, *lifting*) *BACK*[*of ?F2*]
deriv-in-maximal GS MCS-impE mem-Collect-eq)
then obtain *B* **where** *B*: ‹(*G1*,*B*)∈*Chn* ∧ *B* ∈ *MC1* ∧ {⟨*b*⟩*?F2*} ∪ {*f* . □ *f*
∈ *G1*}⊆ *B*›
using *assms GS existenceChn-1*[*of* ⟨*b*⟩*?F2 G1*] **by** *auto*
hence *X3*: ‹{⟨*b*⟩¬ (*As* ⇝ ⊥), *PosAs* ⇝ ⊥} ⊆ *B*› **using** *X1* **by** *auto*
then obtain *C* **where** *C*: ‹*MCS C* ∧ ({¬ (*As* ⇝ ⊥)} ∪ {*f* .[*b*]*f* ∈ *B*}) ⊆ *C*›
using *existenceZmc*[*of* ¬ (*As* ⇝ ⊥) *B*] *B* **by** *auto*
hence ‹(*B*,*C*) ∈ *Zmc*› **using** *B C* **by** *simp*
hence *BC*: ‹({⟨*b*⟩*f* | *f*. *f* ∈ *C*}) ⊆ *B*›
using *pos-b-r2-sub*[*of B C*] *Z-from-MC1-to-MC2*[*of B C*] **by** *simp*
have ‹*set As* ⊆ *C*› **using** *C not-imp-to-conj* **by** *auto*
hence ‹{⟨*b*⟩*f* | *f*. *f* ∈ *set As*} ⊆ *B*› **using** *BC* **by** *auto*
hence ‹*set PosAs* ⊆ *B*› **using** *X* **by** *simp*
hence ‹⊥ ∈ *B*› **using** *B X3*
by (*simp add*: *multiple-MP-MCS*)
thus *False* **using** *B* **by** *simp*
qed
then obtain *G4* **where** *G4*: ‹(*?L* ∪ *?R*) ⊆ *G4* ∧ *MCS G4*›
using *Extend-subset MCS-Extend'* **by** *fastforce*
hence ‹{*B* | *B*. □*B* ∈ *G1*} ⊆ *G4*› **by** *simp*
hence *T1*: ‹(*G1*, *G4*) ∈ *Chn*› **using** *G4 GS* **by** *simp*
hence ‹*G4* ∈ *MC1*› **using** *G4* **by** *auto*
hence ‹(*G4*, *G3*) ∈ *Zmc*› **using** *G4 pos-b-r2-sub GS* **by** *auto*
thus *?thesis* **using** *T1* **by** *auto*
qed

11 Existence of elements in *Zmc*.

lemma *Zmc-not-empty*:
‹*Zmc* ≠ {}›
proof −
let *?V* = ‹λ *w p* . *False*›
let *?M1* = ‹*Model* {*0*::*nat*} {} *?V*›
let *?M2* = ‹*Model* {*0*} {} *?V*›
let *?M* = ‹*ModelLb ?M1 ?M2* {(*0*,*0*)}›
have ‹*bi-model ?M*› **using** *bi-model-def* **by** *force*

have ‹¬ (*?M*, m, *0*) $\models_B$ $\langle b\rangle\top \longrightarrow \bot$› **by** *auto*
hence ‹¬ [$\langle b\rangle\top$] $\vdash_B$ $\bot$› **using** *soundness* ‹*bi-model ?M*› **by** *fastforce*
hence ‹*consistent* {$\langle b\rangle\top$}›
by (*metis K-imply-weaken consistent-def empty-set list.simps*(*15*))
hence ‹$\exists$ *A* . $\langle b\rangle\top \in A \wedge$ *MCS A*›
using *MCS-Extend′ Extend-subset insert-subset* **by** *metis*
hence ‹$\exists$ *A* . $\langle b\rangle\top \in A \wedge A \in$ *MC1*› **by** *auto*
hence ‹$\exists$ *A B* . $\langle b\rangle\top \in A \wedge A \in$ *MC1* $\wedge$ *MCS B* $\wedge$ {*F* . [*b*]*F* $\in A$} $\subseteq B$›
using *existenceZmc* **by** *force*
hence ‹$\exists$ *A B* . (*A*, *B*) $\in$ *Zmc*› **by** *auto*
thus *?thesis* **by** *auto*
qed

12 Canonical Model

The Canonical Model is defined in terms of *MC1*, *MC2*, *Chn* and *Zmc*.

R1 and *R2* are the modal acessibility relations of the models on the left and right of the Canonical Model for $\mathcal{L}_{\Box[b]}$. They are defined as restrictions of *Chn* for *MC1* and *MC2*, respectively. The valuation function *Vc* is common for two models, it assigns True to a variable iff it belongs to the corresponding world. The bisimulation relation *Zc* is defined from *Zmc*.

abbreviation *R1* :: ‹('*p fm set* × '*p fm set*) *set*› **where**
‹*R1* ≡ {(*w1*, *w2*) . *w1* ∈ *MC1* ∧ *w2* ∈ *MC1* ∧ (*w1*, *w2*) ∈ *Chn*}›

abbreviation *R2* :: ‹('*p fm set* × '*p fm set*) *set*› **where**
‹*R2* ≡ {(*w1*, *w2*) . *w1* ∈ *MC2* ∧ *w2* ∈ *MC2* ∧ (*w1*, *w2*) ∈ *Chn*}›

abbreviation *Vc* :: ‹'*p fm set* ⇒ '*p* ⇒ *bool*› **where**
‹*Vc w p* ≡ •*p* ∈ *w*›

abbreviation *Zc* :: ‹('*p fm set* × '*p fm set*) *set*› **where**
‹*Zc* ≡ {(*w1*, *w2*) . *w1* ∈ *MC1* ∧ *w2* ∈ *MC2* ∧ (*w1*, *w2*) ∈ *Zmc*}›

Now, models *Mc1* and *Mc2* are introduced. These are the models on the left and right, of the Canonical Model.

abbreviation *Mc1* :: ‹('*p*, '*p fm set*) *model*› **where**
‹*Mc1* ≡ *Model MC1 R1 Vc*›

abbreviation *Mc2* :: ‹('*p*, '*p fm set*) *model*› **where**
‹*Mc2* ≡ *Model MC2 R2 Vc*›

Finally, the Canonical Model is introduced.

abbreviation *CanMod* :: ‹('*p*, '*p fm set*) *modelLb*› **where**
‹*CanMod* ≡ *ModelLb Mc1 Mc2 Zc*›

lemma *Chn-Rc2*:
‹((*S*,*T*) ∈ *Chn* ∧ *S* ∈ *MC2* ∧ *T* ∈ *MC2*) ⟷ (*S*,*T*) ∈ *R2*› (**is** ‹*?L* ⟷ *?R*›)
proof
assume ‹*?L*›
hence ‹*T* ∈ *MC2*›
by (*metis* (*mono-tags*, *lifting*))
hence ‹*S* ∈ *MC2* ∧ *T* ∈ *MC2*› **using** ‹*?L*› **by** *simp*
thus ‹*?R*› **using** ‹*?L*› **by** *simp*
next
assume ‹*?R*›
thus ‹*?L*› **by** *simp*
qed

13 Canonocity

The Canonical Model is a bi-model.

lemma *bi-model-CM*:
‹*bi-model CanMod*›
proof −
— Properties of Z and W sets
have ‹*Zmc* ≠ {}› **using** *Zmc-not-empty* **by** *simp*
hence ‹∃ *A B* . *A* ∈ *MC1* ∧ *B* ∈ *MC2* ∧ (*A*, *B*) ∈ *Zmc*›
by (*metis* (*mono-tags*, *lifting*) *Z-from-MC1-to-MC2 Collect-empty-eq*
case-prodE mem-Collect-eq)
hence ‹*MC1* ≠ {} ∧ *MC2* ≠ {} ∧ *Zc* ≠ {}›
by (*smt* (*verit*, *ccfv-SIG*) *case-prodE not-empty mem-Collect-eq*
prod.inject split-cong)
hence *WZ*: ‹*W* (*M1 CanMod*) ≠ {} ∧ *W* (*M2 CanMod*) ≠ {} ∧ *Z CanMod* ≠
{} ∧
Z CanMod ⊆ (*W* (*M1 CanMod*)) × (*W* (*M2 CanMod*))› **by** *auto*
— Properties of R1 and R2
have *R1*: ‹*R* (*M1 CanMod*) ⊆ (*W* (*M1 CanMod*)) × (*W* (*M1 CanMod*))› **by**
auto
have *R2*: ‹*R* (*M2 CanMod*) ⊆ (*W* (*M2 CanMod*)) × (*W* (*M2 CanMod*))› **by**
auto
— Atomic Harmony
have ‹∀ *w w′ l* . (*w*, *w′*) ∈ *Z CanMod* ⟶
(•*l* ∈ *w* ⟷ •*l* ∈ *w′*)› **by** (*metis* (*mono-tags*, *lifting*)
Zmc-atomic-harmony mem-Collect-eq modelLb.sel(*3*) *old.prod.case*)
hence *AtHarm*: ‹∀ *w w′* . (*w*, *w′*) ∈ *Z CanMod* ⟶
((*V* (*M1 CanMod*)) *w*) = ((*V* (*M2 CanMod*)) *w′*)› **by** *auto*
— Forth
have *Forth*: ‹∀ *w w′ v* . (*w*, *w′*) ∈ *Z CanMod* ∧ (*w*,*v*) ∈ *R* (*M1 CanMod*) ⟶
(∃ *v′* . (*v*,*v′*) ∈ *Z CanMod* ∧ (*w′*, *v′*) ∈ *R* (*M2 CanMod*))›
by (*smt* (*z3*) *forth-cm Chn-from-to-2 mem-Collect-eq model.sel*(*2*) *modelLb.sel*(*1*,*2*,*3*)
old.prod.case)
— Back
have *Back*: ‹∀ *w w′ v′* . (*w*, *w′*) ∈ *Z CanMod* ∧ (*w′*,*v′*) ∈ *R* (*M2 CanMod*) ⟶

```
    (∃ v . (v,v′) ∈ Z CanMod ∧ (w,v) ∈ R (M1 CanMod))›
  by (smt (z3) back-cm Chn-from-to-2 mem-Collect-eq model.sel(2) modelLb.sel(1,2,3)
      old.prod.case)
 show ?thesis unfolding bi-model-def using WZ R1 R2 AtHarm Forth Back
  by blast
qed
```

14 Truth Lemma

This is the key lemma for Completeness: a formula F is true in a given world w of the Canonical Model iff $F \in w$.

```
lemma Truth-Lemma:
 ‹∀ (S::'p fm set) . (MCS S ⟶ ((S ∈ MC1 ⟶ ((CanMod, m, S)⊨B F ⟷ F
∈ S)) ∧
    (S ∈ MC2 ⟶ ((CanMod, m′, S)⊨B F ⟷ F ∈ S))))› (is ‹?TL F›)
proof (induct F)
 case Fls
 thus ?case by simp
next
 case (Pro x)
 thus ?case by simp
next
 case (Imp F1 F2)
 assume KSq: ‹?TL F1›
 assume Kb: ‹?TL F2›
 have ‹∀ S . MCS S ⟶ (F1 ⟶ F2 ∈ S ⟷ (F1 ∈ S ⟶ F2 ∈ S))› by auto
 thus ?case using KSq Kb by simp
next
 case (Box F)
 assume A: ‹?TL F›
 have B: ‹∀ S . MCS S ∧ S ∈ MC1 ⟶
    ((CanMod, m, S) ⊨B □F ⟷
       (∀ T . (S,T) ∈ R1 ⟶ (CanMod, m, T) ⊨B F))›
  by (smt (z3) mem-Collect-eq model.sel(1,2) modelLb.sel(1) old.prod.case
      semantics.simps(5))
 have B′: ‹∀ S . MCS S ∧ S ∈ MC2 ⟶
    ((CanMod, m′, S) ⊨B □F ⟷
       (∀ T . (S,T) ∈ R2 ⟶ (CanMod, m′, T) ⊨B F))›
  by (smt (z3) mem-Collect-eq model.sel(1,2) modelLb.sel(2) old.prod.case
      semantics.simps(6))
 have C: ‹∀ S . MCS S ∧ S ∈ MC1 ⟶
    (□F ∈ S ⟷ (∀ T . ((S,T) ∈ R1) ⟶ F ∈ T))›
  by (metis (mono-tags, lifting) Chn-iff mem-Collect-eq old.prod.case)
 have C′: ‹∀ S . MCS S ∧ S ∈ MC2 ⟶
    (□F ∈ S ⟷ (∀ T . ((S,T) ∈ R2) ⟶ F ∈ T))›
  by (metis (mono-tags, lifting) Chn-Rc2 Chn-iff Chn-from-to-2 )
 thus ?case using A B B′ C by simp
next
```

case (*FrB F*)
assume *A*: ‹*?TL F*›
have *D1*: ‹∀ *S* . *MCS S* ∧ *S* ∈ *MC1* ⟶ ([*b*]*F* ∈ *S* ⟷
(∀ *T* . ((*S*,*T*) ∈ *Zc*) ⟶ *F* ∈ *T*))›
proof (*intro allI HOL.impI*)
fix *S*
assume *S*: ‹*MCS* (*S*::'*p fm set*) ∧ *S* ∈ *MC1*›
show ‹[*b*]*F* ∈ *S* ⟷ (∀ *T* . ((*S*,*T*) ∈ *Zc*) ⟶ *F* ∈ *T*)›
proof
assume ‹[*b*]*F* ∈ *S*›
thus ‹∀ *T* . ((*S*,*T*) ∈ *Zc*) ⟶ *F* ∈ *T*› **by** *fastforce*
next
assume *T*: ‹∀ *T* . ((*S*,*T*) ∈ *Zc*) ⟶ *F* ∈ *T*›
show ‹[*b*]*F* ∈ *S*›
proof (*rule ccontr*)
assume ‹[*b*]*F* ∉ *S*›
hence ‹⟨*b*⟩¬*F* ∈ *S*›
by (*meson KFrB.not-nec-to-pos-not MCS-imp S deriv-in-maximal*)
hence ‹∃ *B* . *B* ∈ *MC2* ∧ {¬*F*} ∪ {*G* . [*b*]*G* ∈ *S*}⊆*B*›
using *S existenceZmc*[*of* ¬*F S*] **by** *fastforce*
then obtain *B* **where** ‹(*S*,*B*) ∈ *Zmc* ∧ *B* ∈ *MC2* ∧ ¬*F* ∈ *B*›
using *S* **by** *auto*
hence *B'*: ‹(*S*,*B*) ∈ *Zc* ∧ *F* ∉ *B*› **using** *S MCS-imp* **by** *auto*
hence ‹*F* ∈ *B*› **using** *T* **by** *auto*
thus *False* **using** *B'* **by** *simp*
qed
qed
qed
have ‹∀ *S* . *MCS S* ∧ *S* ∈ *MC2* ⟶ [*b*]*F* ∈ *S* ∧ (*CanMod*, m', *S*) ⊨*B* [*b*]*F*›
using *all-box-b-in-MC2* **by** *auto*
thus *?case* **using** *A D1* **by** *auto*
qed

corollary *truth-lemma-MC1*:
assumes ‹*S* ∈ *MC1*›
shows ‹∀ *F* . *F* ∈ *S* ⟷ (*CanMod*, m, *S*) ⊨*B* *F*›
using *assms Truth-Lemma* **by** *auto*

corollary *truth-lemma-MC2*:
assumes ‹*S* ∈ *MC2*›
shows ‹∀ *F* . *F* ∈ *S* ⟷ (*CanMod*, m', *S*) ⊨*B* *F*›
using *assms Truth-Lemma* **by** *auto*

15 Completeness

Proof of strong completeness.

theorem *strong-completeness*:
assumes ‹∀ (*M* :: ('*p*, '*p fm set*) *modelLb*) *ep w* .

```
      (bi-model M ⟶ (
          (w ∈ W (M1 M) ⟶ ((∀ γ ∈ set Γ . (M, m, w) ⊨B γ) ⟶ (M, m, w)
⊨B G)) ∧
          (w ∈ W (M2 M) ⟶ ((∀ γ ∈ set Γ . (M, m′, w) ⊨B γ) ⟶ (M, m′, w)
⊨B G))))›
  shows ‹Γ ⊢B G›
proof (rule ccontr)
  assume ‹¬ (Γ ⊢B G)›
  hence ‹¬ (¬G#Γ ⊢B ⊥)› using K-Boole by blast
  hence ‹consistent (set (¬G#Γ))›
    using K-imply-weaken consistent-underivable by blast
  then obtain S where S: ‹MCS S ∧ set (¬G#Γ) ⊆ S›
    using Extend-subset MCS-Extend′ by blast
  have ‹S ∈ MC1› using S by auto
  hence N: ‹(∀ F ∈ set Γ . (CanMod, m, S) ⊨B F) ∧ ((CanMod, m, S) ⊨B ¬G)›

    using S ‹S ∈ MC1› Truth-Lemma
    by (smt (z3) insert-subset list.simps(15) subsetD)
  hence ‹(CanMod, m, S) ⊨B G› using bi-model-CM ‹S ∈ MC1› assms by auto
  thus False using N semantics.simps by auto
qed
```

Definition of validity in bi-models:

```
abbreviation bi-model-valid :: ‹'p fm ⇒ bool› where
  ‹bi-model-valid p ≡ ∀ (M :: ('p, 'p fm set) modelLb) w. bi-model M ⟶
      ((w ∈ W (M1 M) ⟶ (M, m, w) ⊨B p) ∧
       (w ∈ W (M2 M) ⟶ (M, m′, w) ⊨B p))›
```

Weak completeness and main result:

```
corollary completeness: ‹bi-model-valid p ⟹ ⊢B p›
  by (metis imply.simps(1) strong-completeness)

theorem main: ‹(bi-model-valid p) ⟷ ⊢B p›
  using soundness completeness by metis
```

16 Extension of atomic harmony to all formulas in $\mathcal{L}_\Box$

Set of formulas in $\mathcal{L}_\Box$.

```
inductive-set Lbox :: ‹'p fm set› where
  Fls: ‹⊥ ∈ Lbox›
| Pro: ‹•l ∈ Lbox›
| Imp: ‹A ∈ Lbox ⟹ B ∈ Lbox ⟹ A⟶B ∈ Lbox›
| Box:   ‹A ∈ Lbox ⟹ □A ∈ Lbox›
```

Auxiliary lemmas for the induction.

```
lemma BotPos:
```

shows ‹$\vdash_B \bot \longrightarrow [b]\bot$› **using** *TAU* **by** *force*

lemma *BotNeg*:
shows ‹$\vdash_B \neg\bot \longrightarrow [b]\neg\bot$›
by (*metis K-imply-Cons K-imply-head Nb imply.simps*(*1*,*2*))

lemma *impPos*:
assumes ‹$\vdash_B \neg A \longrightarrow [b]\neg A$›
and ‹$\vdash_B B \longrightarrow [b]B$›
shows ‹$\vdash_B (A \longrightarrow B) \longrightarrow [b](A \longrightarrow B)$›
proof −
have ‹$\vdash_B \neg A \longrightarrow (A \longrightarrow B)$› **using** *TAU* **by** *force*
hence ‹$\vdash_B [b]\neg A \longrightarrow [b](A \longrightarrow B)$› **using** *Nb Kb MP* **by** *force*
hence *1*: ‹$\vdash_B \neg A \longrightarrow [b](A \longrightarrow B)$› **using** *assms*(*1*) *MP-chain* **by** *auto*
have ‹$\vdash_B B \longrightarrow (A \longrightarrow B)$› **using** *TAU* **by** *force*
hence ‹$\vdash_B [b]B \longrightarrow [b](A \longrightarrow B)$› **using** *Nb Kb MP* **by** *force*
hence *2*: ‹$\vdash_B B \longrightarrow [b](A \longrightarrow B)$› **using** *assms*(*2*) *MP-chain* **by** *auto*
have ‹$\vdash_B (A \longrightarrow B) \longrightarrow (\neg A \longrightarrow [b](A \longrightarrow B)) \longrightarrow$
$(B \longrightarrow [b](A \longrightarrow B)) \longrightarrow [b](A \longrightarrow B)$› **using** *TAU* **by** *force*
thus *?thesis* **using** *1 2 MPcons* **by** *blast*
qed

lemma *impNeg*:
assumes ‹$\vdash_B A \longrightarrow [b]A$›
and ‹$\vdash_B \neg B \longrightarrow [b]\neg B$›
shows ‹$\vdash_B \neg(A \longrightarrow B) \longrightarrow [b]\neg(A \longrightarrow B)$›
proof −
have *1*: ‹$\vdash_B \neg(A \longrightarrow B) \longrightarrow A \wedge \neg B$› **using** *TAU* **by** *force*
have ‹$\vdash_B (A \wedge \neg B) \longrightarrow (A \longrightarrow [b]A) \longrightarrow (\neg B \longrightarrow [b]\neg B) \longrightarrow [b]A \wedge [b]\neg B$›
using *TAU* **by** *force*
hence *2*: ‹$\vdash_B \neg(A \longrightarrow B) \longrightarrow [b]A \wedge [b]\neg B$› **using** *1 MP-chain MPcons assms*
by *blast*
have ‹$\vdash_B A \longrightarrow \neg B \longrightarrow \neg(A \longrightarrow B)$› **using** *TAU* **by** *force*
hence *3*: ‹$\vdash_B [b]A \longrightarrow [b]\neg B \longrightarrow [b]\neg(A \longrightarrow B)$› **by** (*metis MP-chain Nb MP*
Kb)
have ‹$\vdash_B ([b]A \longrightarrow [b]\neg B \longrightarrow [b]\neg(A \longrightarrow B)) \longrightarrow [b]A \wedge [b]\neg B \longrightarrow [b]\neg($*A*
$\longrightarrow B)$›
using *TAU* **by** *force*
hence *4*: ‹$\vdash_B [b]A \wedge [b]\neg B \longrightarrow [b]\neg(A \longrightarrow B)$› **using** *3 MP* **by** *force*
thus *?thesis* **using** *2 MP-chain* **by** *auto*
qed

lemma *NSqPos*:
assumes ‹$\vdash_B A \longrightarrow [b]A$›
shows ‹$\vdash_B \Box A \longrightarrow [b]\Box A$›
using *BACK-rev KSq MP MP-chain NSq assms* **by** *blast*

lemma *NSqNeg*:

assumes ‹$\vdash_B \neg A \longrightarrow [b]\neg A$›
shows ‹$\vdash_B \neg\Box A \longrightarrow [b]\neg\Box A$›
proof −
have *1*: ‹$\vdash_B \neg\Box A \longrightarrow \Diamond\neg A$›
by (*simp add*: *KBox.not-nec-to-pos-not*)
have *2*: ‹$\vdash_B \Diamond\neg A \longrightarrow \Diamond[b]\neg A$› **using** *assms KBox.KN KBox.Kpos MP* **by** *blast*
have *3*: ‹$\vdash_B (\neg[b]\neg\Box A \wedge \Diamond[b]\neg A \longrightarrow \bot) \longrightarrow (\Diamond[b]\neg A \longrightarrow [b]\neg\Box A)$› **using** *TAU* **by** *force*
have *4*: ‹$\vdash_B \neg[b]\neg\Box A \wedge \Diamond[b]\neg A \longrightarrow \langle b\rangle(\Box A \wedge \Diamond\neg A)$› **using** *FORTH* **by** *force*
have *5*: ‹$\vdash_B \Diamond\neg A \longrightarrow \neg\Box A$› **by** (*simp add*: *KBox.pos-not-to-not-nec*)
have ‹$\vdash_B (\Diamond\neg A \longrightarrow \neg\Box A) \longrightarrow \neg(\Box A \wedge \Diamond\neg A)$› **using** *TAU* **by** *force*
hence ‹$\vdash_B \neg(\Box A \wedge \Diamond\neg A)$› **using** *5 MP* **by** *auto*
hence ‹$\vdash_B [b]\neg(\Box A \wedge \Diamond\neg A)$› **using** *Nb* **by** *blast*
hence *6*: ‹$\vdash_B \langle b\rangle(\Box A \wedge \Diamond\neg A) \longrightarrow \bot$›
by (*meson KFrB.not-nec-to-pos-not KFrB.not-pos-to-nec-not KFrB.pos-not-to-not-nec MP MPcons*)
hence ‹$\vdash_B \neg[b]\neg\Box A \wedge \Diamond[b]\neg A \longrightarrow \bot$› **using** *4 MP-chain* **by** *blast*
hence ‹$\vdash_B \Diamond[b]\neg A \longrightarrow [b]\neg\Box A$› **using** *3 MP* **by** *blast*
thus *?thesis* **using** *1 2 MP-chain* **by** *blast*
qed

The following lemma extends atomic harmony (HARM) to all formulas in $\mathcal{L}_\Box$.

lemma *Lbox-harm*:
assumes ‹$A \in$ *Lbox*›
shows‹$\vdash_B A \longrightarrow [b]A$›
proof −
have ‹$\vdash_B A \longrightarrow [b]A \wedge \vdash_B \neg A \longrightarrow [b]\neg A$›
using *assms*
proof (*induct A rule*: *Lbox.induct*)
case *Fls*
thus *?case* **using** *BotPos BotNeg* **by** *simp*
next
case (*Pro l*)
thus *?case* **using** *HARM* **by** *fastforce*
next
case (*Imp A B*)
thus *?case* **using** *impPos impNeg* **by** *fastforce*
next
case (*Box A*)
thus *?case* **using** *NSqPos NSqNeg* **by** *fastforce*
qed
thus *?thesis* **by** *simp*
qed

end

References

[1] A. Burrieza, F. Soler-Toscano, and A. Yuste-Ginel. A meta-modal logic for bisimulations, 2025. https://arxiv.org/abs/2507.15117.

[2] A. H. From. Synthetic completeness. *Archive of Formal Proofs*, January 2023. https://www.isa-afp.org/entries/Synthetic_Completeness.html, Formal proof development.