Public Announcement Logic

Asta Halkjær From

December 14, 2021

Abstract

This work is a formalization of public announcement logic with countably many agents. It includes proofs of soundness and completeness for a variant of the axiom system PA + DIST! + NEC! [1]. The completeness proof builds on the Epistemic Logic theory.

Contents

theory *PAL* **imports** *Epistemic-Logic.Epistemic-Logic* **begin**

1 Syntax

datatype *'i pfm*
= *FF* ($\bot_!$)
| *Pro'* *id* (*Pro*$_!$)
| *Dis* ‹*'i pfm*› ‹*'i pfm*› (**infixr** $\vee_!$ *30*)
| *Con* ‹*'i pfm*› ‹*'i pfm*› (**infixr** $\wedge_!$ *35*)
| *Imp* ‹*'i pfm*› ‹*'i pfm*› (**infixr** $\longrightarrow_!$ *25*)
| *K'* *'i* ‹*'i pfm*› (*K*$_!$)
| *Ann* ‹*'i pfm*› ‹*'i pfm*› ([-]$_!$ - [*50*, *50*] *50*)

abbreviation *PIff* :: ‹*'i pfm* ⇒ *'i pfm* ⇒ *'i pfm*› (**infixr** $\longleftrightarrow_!$ *25*) **where**
‹$p \longleftrightarrow_! q \equiv (p \longrightarrow_! q) \wedge_! (q \longrightarrow_! p)$›

2 Semantics

fun
psemantics :: ‹(*'i*, *'w*) *kripke* ⇒ *'w* ⇒ *'i pfm* ⇒ *bool*› (-, - $\models_!$ - [*50*, *50*] *50*) **and**
restrict :: ‹(*'i*, *'w*) *kripke* ⇒ *'i pfm* ⇒ (*'i*, *'w*) *kripke*› **where**
‹$(M, w \models_! \bot_!) = False$›
| ‹$(M, w \models_! Pro_! x) = \pi\ M\ w\ x$›
| ‹$(M, w \models_! (p \vee_! q)) = ((M, w \models_! p) \vee (M, w \models_! q))$›
| ‹$(M, w \models_! (p \wedge_! q)) = ((M, w \models_! p) \wedge (M, w \models_! q))$›
| ‹$(M, w \models_! (p \longrightarrow_! q)) = ((M, w \models_! p) \longrightarrow (M, w \models_! q))$›
| ‹$(M, w \models_! K_!\ i\ p) = (\forall v \in \mathcal{W}\ M \cap \mathcal{K}\ M\ i\ w.\ M, v \models_! p)$›
| ‹$(M, w \models_! [r]_!\ p) = ((M, w \models_! r) \longrightarrow (restrict\ M\ r, w \models_! p))$›
| ‹$restrict\ M\ p = Kripke\ \{w\ |w.\ w \in \mathcal{W}\ M \wedge M, w \models_! p\}\ (\pi\ M)\ (\mathcal{K}\ M)$›

primrec *static* :: ‹*'i pfm* ⇒ *bool*› **where**
‹$static\ \bot_! = True$›
| ‹$static\ (Pro_!\ \text{-}) = True$›
| ‹$static\ (p \vee_! q) = (static\ p \wedge static\ q)$›
| ‹$static\ (p \wedge_! q) = (static\ p \wedge static\ q)$›
| ‹$static\ (p \longrightarrow_! q) = (static\ p \wedge static\ q)$›
| ‹$static\ (K_!\ i\ p) = static\ p$›
| ‹$static\ ([r]_!\ p) = False$›

primrec *lower* :: ‹*'i pfm* ⇒ *'i fm*› **where**
‹$lower\ \bot_! = \bot$›
| ‹$lower\ (Pro_!\ x) = Pro\ x$›
| ‹$lower\ (p \vee_! q) = (lower\ p \vee lower\ q)$›
| ‹$lower\ (p \wedge_! q) = (lower\ p \wedge lower\ q)$›
| ‹$lower\ (p \longrightarrow_! q) = (lower\ p \longrightarrow lower\ q)$›
| ‹$lower\ (K_!\ i\ p) = K\ i\ (lower\ p)$›

| ‹*lower* ($[r]_!$ *p*) = *undefined*›

primrec *lift* :: ‹'*i fm* ⇒ '*i pfm*› **where**
  ‹*lift* ⊥ = $\bot_!$›
| ‹*lift* (*Pro x*) = $Pro_!$ *x*›
| ‹*lift* (*p* ∨ *q*) = (*lift p* $\vee_!$ *lift q*)›
| ‹*lift* (*p* ∧ *q*) = (*lift p* $\wedge_!$ *lift q*)›
| ‹*lift* (*p* ⟶ *q*) = (*lift p* $\longrightarrow_!$ *lift q*)›
| ‹*lift* (*K i p*) = $K_!$ *i* (*lift p*)›

lemma *lower-semantics*:
  assumes ‹*static p*›
  shows ‹(*M*, *w* ⊨ *lower p*) ⟷ (*M*, *w* $\models_!$ *p*)›
  using *assms* **by** (*induct p arbitrary*: *w*) *simp-all*

lemma *lift-semantics*: ‹(*M*, *w* ⊨ *p*) ⟷ (*M*, *w* $\models_!$ *lift p*)›
  by (*induct p arbitrary*: *w*) *simp-all*

lemma *lower-lift*: ‹*lower* (*lift p*) = *p*›
  by (*induct p*) *simp-all*

lemma *lift-lower*: ‹*static p* ⟹ *lift* (*lower p*) = *p*›
  by (*induct p*) *simp-all*

3 Soundness of Reduction

primrec *reduce'* :: ‹'*i pfm* ⇒ '*i pfm* ⇒ '*i pfm*› **where**
  ‹*reduce'* *r* $\bot_!$ = (*r* $\longrightarrow_!$ $\bot_!$)›
| ‹*reduce'* *r* ($Pro_!$ *x*) = (*r* $\longrightarrow_!$ $Pro_!$ *x*)›
| ‹*reduce'* *r* (*p* $\vee_!$ *q*) = (*reduce' r p* $\vee_!$ *reduce' r q*)›
| ‹*reduce'* *r* (*p* $\wedge_!$ *q*) = (*reduce' r p* $\wedge_!$ *reduce' r q*)›
| ‹*reduce'* *r* (*p* $\longrightarrow_!$ *q*) = (*reduce' r p* $\longrightarrow_!$ *reduce' r q*)›
| ‹*reduce'* *r* ($K_!$ *i p*) = (*r* $\longrightarrow_!$ $K_!$ *i* (*reduce' r p*))›
| ‹*reduce'* *r* ($[p]_!$ *q*) = *undefined*›

primrec *reduce* :: ‹'*i pfm* ⇒ '*i pfm*› **where**
  ‹*reduce* $\bot_!$ = $\bot_!$›
| ‹*reduce* ($Pro_!$ *x*) = $Pro_!$ *x*›
| ‹*reduce* (*p* $\vee_!$ *q*) = (*reduce p* $\vee_!$ *reduce q*)›
| ‹*reduce* (*p* $\wedge_!$ *q*) = (*reduce p* $\wedge_!$ *reduce q*)›
| ‹*reduce* (*p* $\longrightarrow_!$ *q*) = (*reduce p* $\longrightarrow_!$ *reduce q*)›
| ‹*reduce* ($K_!$ *i p*) = $K_!$ *i* (*reduce p*)›
| ‹*reduce* ($[r]_!$ *p*) = *reduce'* (*reduce r*) (*reduce p*)›

lemma *static-reduce'*: ‹*static p* ⟹ *static r* ⟹ *static* (*reduce' r p*)›
  by (*induct p*) *simp-all*

lemma *static-reduce*: ‹*static* (*reduce p*)›
  by (*induct p*) (*simp-all add*: *static-reduce'*)

lemma *reduce′-semantics*:
assumes ‹*static q*›
shows ‹((*M*, *w* ⊨! [*p*]! (*q*))) = (*M*, *w* ⊨! *reduce′ p q*)›
using *assms* **by** (*induct q arbitrary*: *w*) *auto*

lemma *reduce-semantics*: ‹(*M*, *w* ⊨! *p*) = (*M*, *w* ⊨! *reduce p*)›
proof (*induct p arbitrary*: *M w*)
case (*Ann p q*)
then show *?case*
using *reduce′-semantics static-reduce* **by** *fastforce*
qed *simp-all*

4 Proof System

primrec *peval* :: ‹(*id* ⇒ *bool*) ⇒ (*′i pfm* ⇒ *bool*) ⇒ *′i pfm* ⇒ *bool*› **where**
‹*peval* - - ⊥! = *False*›
| ‹*peval g* - (*Pro*! *x*) = *g x*›
| ‹*peval g h* (*p* ∨! *q*) = (*peval g h p* ∨ *peval g h q*)›
| ‹*peval g h* (*p* ∧! *q*) = (*peval g h p* ∧ *peval g h q*)›
| ‹*peval g h* (*p* ⟶! *q*) = (*peval g h p* ⟶ *peval g h q*)›
| ‹*peval* - *h* (*K*! *i p*) = *h* (*K*! *i p*)›
| ‹*peval* - *h* ([*r*]! *p*) = *h* ([*r*]! *p*)›

abbreviation ‹*ptautology p* ≡ ∀ *g h*. *peval g h p*›

inductive *PAK* :: ‹(*′i pfm* ⇒ *bool*) ⇒ *′i pfm* ⇒ *bool*› (- ⊢! - [*50*, *50*] *50*)
for *A* :: ‹*′i pfm* ⇒ *bool*› **where**
PA1: ‹*ptautology p* ⟹ *A* ⊢! *p*›
| *PA2*: ‹*A* ⊢! (*K*! *i p* ∧! *K*! *i* (*p* ⟶! *q*) ⟶! *K*! *i q*)›
| *PAx*: ‹*A p* ⟹ *A* ⊢! *p*›
| *PR1*: ‹*A* ⊢! *p* ⟹ *A* ⊢! (*p* ⟶! *q*) ⟹ *A* ⊢! *q*›
| *PR2*: ‹*A* ⊢! *p* ⟹ *A* ⊢! *K*! *i p*›
| *PFF*: ‹*A* ⊢! ([*r*]! ⊥! ⟷! (*r* ⟶! ⊥!))›
| *PPro*: ‹*A* ⊢! ([*r*]! *Pro*! *x* ⟷! (*r* ⟶! *Pro*! *x*))›
| *PDis*: ‹*A* ⊢! ([*r*]! (*p* ∨! *q*) ⟷! [*r*]! *p* ∨! [*r*]! *q*)›
| *PCon*: ‹*A* ⊢! ([*r*]! (*p* ∧! *q*) ⟷! [*r*]! *p* ∧! [*r*]! *q*)›
| *PImp*: ‹*A* ⊢! (([*r*]! (*p* ⟶! *q*)) ⟷! ([*r*]! *p* ⟶! [*r*]! *q*))›
| *PK*: ‹*A* ⊢! (([*r*]! *K*! *i p*) ⟷! (*r* ⟶! *K*! *i* ([*r*]! *p*)))›
| *PAnn*: ‹*A* ⊢! *p* ⟹ *A* ⊢! [*r*]! *p*›

lemma *eval-peval*: ‹*eval h* (*g o lift*) *p* = *peval h g* (*lift p*)›
by (*induct p*) *simp-all*

lemma *tautology-ptautology*: ‹*tautology p* ⟹ *ptautology* (*lift p*)›
using *eval-peval* **by** *blast*

lemma *peval-eval*:
assumes ‹*static p*›

shows ‹*eval h g* (*lower p*) = *peval h* (*g o lower*) *p*›
using *assms* **by** (*induct p*) *simp-all*

lemma *ptautology-tautology*:
assumes ‹*static p*›
shows ‹*ptautology p* ⟹ *tautology* (*lower p*)›
using *assms peval-eval* **by** *blast*

theorem *AK-PAK*: ‹*A o lift* ⊢ *p* ⟹ *A* ⊢! *lift p*›
by (*induct p rule*: *AK.induct*) (*auto intro*: *PAK.intros*(*1*−*5*) *simp*: *tautology-ptautology*)

theorem *static-completeness*:
assumes ‹*static p*› ‹∀(*M* :: ('*i* :: *countable*, '*i fm set*) *kripke*) *w*. *M*, *w* ⊨! *p*›
shows ‹*A* ⊢! *p*›
proof −
have ‹∀(*M* :: ('*i* :: *countable*, '*i fm set*) *kripke*) *w*. *M*, *w* ⊨ *lower p*›
using *assms* **by** (*simp add*: *lower-semantics*)
then have ‹*A o lift* ⊢ *lower p*›
by (*simp add*: *completeness*)
then have ‹*A* ⊢! *lift* (*lower p*)›
using *AK-PAK* **by** *fast*
then show *?thesis*
using *assms*(*1*) *lift-lower* **by** *metis*
qed

5 Soundness

lemma *peval-semantics*: ‹*peval* (*val w*) (λ*q*. *Kripke W val r*, *w* ⊨! *q*) *p* = (*Kripke W val r*, *w* ⊨! *p*)›
by (*induct p*) *simp-all*

lemma *ptautology*:
assumes ‹*ptautology p*›
shows ‹*M*, *w* ⊨! *p*›
proof −
from *assms* **have** ‹*peval* (*g w*) (λ*q*. *Kripke W g r*, *w* ⊨! *q*) *p*› **for** *W g r*
by *simp*
then have ‹*Kripke W g r*, *w* ⊨! *p*› **for** *W g r*
using *peval-semantics* **by** *fast*
then show ‹*M*, *w* ⊨! *p*›
by (*metis kripke.collapse*)
qed

theorem *soundness*:
fixes *M* :: ‹('*i*, '*w*) *kripke*›
assumes
‹⋀(*M* :: ('*i*, '*w*) *kripke*) *w p*. *A p* ⟹ *P M* ⟹ *M*, *w* ⊨! *p*›
‹⋀(*M* :: ('*i*, '*w*) *kripke*) *p*. *P M* ⟹ *P* (*restrict M p*)›
shows ‹*A* ⊢! *p* ⟹ *P M* ⟹ *M*, *w* ⊨! *p*›

proof (*induct p arbitrary*: *M w rule*: *PAK.induct*)
case (*PAnn p r*)
moreover have ‹*P* (*restrict M r*)›
using *PAnn*(*3*) *assms*(*2*) **by** *simp*
ultimately show *?case*
by *simp*
qed (*simp-all add*: *assms ptautology*)

corollary ‹(λ-. *False*) ⊢! *p* ⟹ *M*, *w* ⊨! *p*›
using *soundness*[**where** *P*=‹λ-. *True*›] **by** *metis*

6 Completeness

lemma *ConE*:
assumes ‹*A* ⊢! (*p* ∧! *q*)›
shows ‹*A* ⊢! *p*› ‹*A* ⊢! *q*›
using *assms* **by** (*metis PA1 PR1 peval.simps*(*4*−*5*))+

lemma *Iff-Dis*:
assumes ‹*A* ⊢! (*p* ⟷! *p*′)› ‹*A* ⊢! (*q* ⟷! *q*′)›
shows ‹*A* ⊢! ((*p* ∨! *q*) ⟷! (*p*′ ∨! *q*′))›
proof −
have ‹*A* ⊢! ((*p* ⟷! *p*′) ⟶! (*q* ⟷! *q*′) ⟶! ((*p* ∨! *q*) ⟷! (*p*′ ∨! *q*′)))›
by (*simp add*: *PA1*)
then show *?thesis*
using *assms PR1* **by** *blast*
qed

lemma *Iff-Con*:
assumes ‹*A* ⊢! (*p* ⟷! *p*′)› ‹*A* ⊢! (*q* ⟷! *q*′)›
shows ‹*A* ⊢! ((*p* ∧! *q*) ⟷! (*p*′ ∧! *q*′))›
proof −
have ‹*A* ⊢! ((*p* ⟷! *p*′) ⟶! (*q* ⟷! *q*′) ⟶! ((*p* ∧! *q*) ⟷! (*p*′ ∧! *q*′)))›
by (*simp add*: *PA1*)
then show *?thesis*
using *assms PR1* **by** *blast*
qed

lemma *Iff-Imp*:
assumes ‹*A* ⊢! (*p* ⟷! *p*′)› ‹*A* ⊢! (*q* ⟷! *q*′)›
shows ‹*A* ⊢! ((*p* ⟶! *q*) ⟷! (*p*′ ⟶! *q*′))›
proof −
have ‹*A* ⊢! ((*p* ⟷! *p*′) ⟶! (*q* ⟷! *q*′) ⟶! ((*p* ⟶! *q*) ⟷! (*p*′ ⟶! *q*′)))›
by (*simp add*: *PA1*)
then show *?thesis*
using *assms PR1* **by** *blast*
qed

lemma *Iff-sym*: ‹(*A* ⊢! (*p* ⟷! *q*)) = (*A* ⊢! (*q* ⟷! *p*))›

proof −
  have ‹$A \vdash_! ((p \longleftrightarrow_! q) \longleftrightarrow_! (q \longleftrightarrow_! p))$›
    by (*simp add*: *PA1*)
  then show *?thesis*
    using *PR1 ConE* **by** *blast*
qed

lemma *Iff-Iff*:
  assumes ‹$A \vdash_! (p \longleftrightarrow_! p')$› ‹$A \vdash_! (p \longleftrightarrow_! q)$›
  shows ‹$A \vdash_! (p' \longleftrightarrow_! q)$›
proof −
  have ‹*ptautology* $((p \longleftrightarrow_! p') \longrightarrow_! (p \longleftrightarrow_! q) \longrightarrow_! (p' \longleftrightarrow_! q))$›
    by (*metis peval.simps*(*4*−*5*))
  with *PA1* **have** ‹$A \vdash_! ((p \longleftrightarrow_! p') \longrightarrow_! (p \longleftrightarrow_! q) \longrightarrow_! (p' \longleftrightarrow_! q))$› **.**
  then show *?thesis*
    using *assms PR1* **by** *blast*
qed

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 *PA2* **by** *fast*
  moreover have ‹$A \vdash_! ((P \wedge_! Q \longrightarrow_! R) \longrightarrow_! (Q \longrightarrow_! P \longrightarrow_! R))$› **for** *P Q R*
    by (*simp add*: *PA1*)
  ultimately show *?thesis*
    using *PR1* **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 *PR2* **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 *PR1* **by** *fast*
qed

lemma *ConI*:
  assumes ‹$A \vdash_! p$› ‹$A \vdash_! q$›
  shows ‹$A \vdash_! (p \wedge_! q)$›
proof −
  have ‹$A \vdash_! (p \longrightarrow_! q \longrightarrow_! p \wedge_! q)$›
    by (*simp add*: *PA1*)
  then show *?thesis*
    using *assms PR1* **by** *blast*

qed

lemma *Iff-wk*:
assumes ‹$A \vdash_! (p \longleftrightarrow_! q)$›
shows ‹$A \vdash_! ((r \longrightarrow_! p) \longleftrightarrow_! (r \longrightarrow_! q))$›
proof −
have ‹$A \vdash_! ((p \longleftrightarrow_! q) \longrightarrow_! ((r \longrightarrow_! p) \longleftrightarrow_! (r \longrightarrow_! q)))$›
by (*simp add*: *PA1*)
then show *?thesis*
using *assms PR1* **by** *blast*
qed

lemma *Iff-reduce'*:
assumes ‹*static p*›
shows ‹$A \vdash_! ([r]_! \ p \longleftrightarrow_! \mathit{reduce'}\ r\ p)$›
using *assms*
proof (*induct p rule*: *pfm.induct*)
case *FF*
then show *?case*
by (*simp add*: *PFF*)
next
case (*Pro' x*)
then show *?case*
by (*simp add*: *PPro*)
next
case (*Dis p q*)
then have ‹$A \vdash_! ([r]_! \ p \vee_! [r]_! \ q \longleftrightarrow_! \mathit{reduce'}\ r\ (p \vee_! q))$›
using *Iff-Dis* **by** *force*
moreover have ‹$A \vdash_! (([r]_! \ p \vee_! [r]_! \ q) \longleftrightarrow_! ([r]_! \ (p \vee_! q)))$›
using *PDis Iff-sym* **by** *fastforce*
ultimately show *?case*
using *PA1 PR1 Iff-Iff* **by** *blast*
next
case (*Con p q*)
then have ‹$A \vdash_! ([r]_! \ p \wedge_! [r]_! \ q \longleftrightarrow_! \mathit{reduce'}\ r\ (p \wedge_! q))$›
using *Iff-Con* **by** *force*
moreover have ‹$A \vdash_! (([r]_! \ p \wedge_! [r]_! \ q) \longleftrightarrow_! ([r]_! \ (p \wedge_! q)))$›
using *PCon Iff-sym* **by** *fastforce*
ultimately show *?case*
using *PA1 PR1 Iff-Iff* **by** *blast*
next
case (*Imp p q*)
then have ‹$A \vdash_! (([r]_! \ p \longrightarrow_! [r]_! \ q) \longleftrightarrow_! \mathit{reduce'}\ r\ (p \longrightarrow_! q))$›
using *Iff-Imp* **by** *force*
moreover have ‹$A \vdash_! (([r]_! \ p \longrightarrow_! [r]_! \ q) \longleftrightarrow_! ([r]_! \ (p \longrightarrow_! q)))$›
using *PImp Iff-sym* **by** *fastforce*
ultimately show *?case*
using *PA1 PR1 Iff-Iff* **by** *blast*
next

```
  case (K' i p)
  then have ‹A ⊢! ([r]! p ⟷! reduce' r p)›
    by simp
  then have ‹A ⊢! (K! i ([r]! p) ⟷! K! i (reduce' r p))›
    using K'-map ConE ConI by metis
  moreover have ‹A ⊢! ([r]! K! i p ⟷! r ⟶! K! i ([r]! p))›
    using PK .
  ultimately have ‹A ⊢! ([r]! K! i p ⟷! r ⟶! K! i (reduce' r p))›
    by (meson Iff-Iff Iff-sym Iff-wk)
  then show ?case
    by simp
next
  case (Ann r p)
  then show ?case
    by simp
qed

lemma Iff-Ann1:
  assumes r: ‹A ⊢! (r ⟷! r')› and ‹static p›
  shows ‹A ⊢! ([r]! p ⟷! [r']! p)›
  using assms(2−)
proof (induct p)
  case FF
  have ‹A ⊢! ((r ⟷! r') ⟶! ((r ⟶! ⊥!) ⟷! (r' ⟶! ⊥!)))›
    by (auto intro: PA1)
  then have ‹A ⊢! ((r ⟶! ⊥!) ⟷! (r' ⟶! ⊥!))›
    using r PR1 by blast
  then show ?case
    by (meson PFF Iff-Iff Iff-sym)
next
  case (Pro' x)
  have ‹A ⊢! ((r ⟷! r') ⟶! ((r ⟶! Pro! x) ⟷! (r' ⟶! Pro! x)))›
    by (auto intro: PA1)
  then have ‹A ⊢! ((r ⟶! Pro! x) ⟷! (r' ⟶! Pro! x))›
    using r PR1 by blast
  then show ?case
    by (meson PPro Iff-Iff Iff-sym)
next
  case (Dis p q)
  then have ‹A ⊢! ([r]! p ∨! [r]! q ⟷! [r']! p ∨! [r']! q)›
    by (simp add: Iff-Dis)
  then show ?case
    by (meson PDis Iff-Iff Iff-sym)
next
  case (Con p q)
  then have ‹A ⊢! ([r]! p ∧! [r]! q ⟷! [r']! p ∧! [r']! q)›
    by (simp add: Iff-Con)
  then show ?case
    by (meson PCon Iff-Iff Iff-sym)
```

next
case (*Imp p q*)
then have ‹$A \vdash_! ((\lfloor r \rfloor_! \ p \longrightarrow_! \lfloor r \rfloor_! \ q) \longleftrightarrow_! (\lfloor r' \rfloor_! \ p \longrightarrow_! \lfloor r' \rfloor_! \ q))$›
by (*simp add*: *Iff-Imp*)
then show *?case*
by (*meson PImp Iff-Iff Iff-sym*)
next
case (*K' i p*)
then have ‹$A \vdash_! (\lfloor r \rfloor_! \ p \longleftrightarrow_! \lfloor r' \rfloor_! \ p)$›
by *simp*
then have ‹$A \vdash_! (K_! \ i \ (\lfloor r \rfloor_! \ p) \longleftrightarrow_! K_! \ i \ (\lfloor r' \rfloor_! \ p))$›
using *K'-map ConE ConI* **by** *metis*
then show *?case*
by (*meson Iff-Iff Iff-Imp Iff-sym PK r*)
next
case (*Ann s p*)
then show *?case*
by *simp*
qed

lemma *Iff-Ann2*:
assumes ‹$A \vdash_! (p \longleftrightarrow_! p')$›
shows ‹$A \vdash_! (\lfloor r \rfloor_! \ p \longleftrightarrow_! \lfloor r \rfloor_! \ p')$›
using *assms PAnn ConE ConI PImp PR1* **by** *metis*

lemma *Iff-reduce*: ‹$A \vdash_! (p \longleftrightarrow_! reduce \ p)$›
proof (*induct p*)
case (*Dis p q*)
then show *?case*
by (*simp add*: *Iff-Dis*)
next
case (*Con p q*)
then show *?case*
by (*simp add*: *Iff-Con*)
next
case (*Imp p q*)
then show *?case*
by (*simp add*: *Iff-Imp*)
next
case (*K' i p*)
have
‹$A \vdash_! (K_! \ i \ p \longrightarrow_! K_! \ i \ (reduce \ p))$›
‹$A \vdash_! (K_! \ i \ (reduce \ p) \longrightarrow_! K_! \ i \ p)$›
using *K' K'-map ConE* **by** *fast+*
then have ‹$A \vdash_! (K_! \ i \ p \longleftrightarrow_! K_! \ i \ (reduce \ p))$›
using *ConI* **by** *blast*
then show *?case*
by *simp*
next

```
  case (Ann r p)
  have ‹A ⊢! ([reduce r]! reduce p ⟷! reduce' (reduce r) (reduce p))›
    using Iff-reduce' static-reduce by blast
  moreover have ‹A ⊢! ([r]! reduce p ⟷! [reduce r]! reduce p)›
    using Ann(1) Iff-Ann1 static-reduce by blast
  ultimately have ‹A ⊢! ([r]! reduce p ⟷! reduce' (reduce r) (reduce p))›
    using Iff-Iff Iff-sym by blast
  moreover have ‹A ⊢! ([r]! reduce p ⟷! [r]! p)›
    by (meson Ann(2) static-reduce Iff-Ann2 Iff-sym)
  ultimately have ‹A ⊢! ([r]! p ⟷! reduce' (reduce r) (reduce p))›
    using Iff-Iff by blast
  then show ?case
    by simp
qed (simp-all add: PA1)

theorem completeness:
  assumes ‹∀(M :: ('i :: countable, 'i fm set) kripke) w. M, w ⊨! p›
  shows ‹A ⊢! p›
proof −
  have ‹∀(M :: ('i :: countable, 'i fm set) kripke) w. M, w ⊨! reduce p›
    using assms reduce-semantics by fast
  moreover have ‹static (reduce p)›
    using static-reduce by fast
  ultimately have ‹A ⊢! reduce p›
    using static-completeness by blast
  moreover have ‹A ⊢! (p ⟷! reduce p)›
    using Iff-reduce by blast
  ultimately show ?thesis
    using ConE(2) PR1 by blast
qed

end
```

References

[1] Y. Wang and Q. Cao. On axiomatizations of public announcement logic. *Synthese*, 190(Supplement-1):103–134, 2013.