```
theory Core_Logic imports Main begin

datatype form
  = Pro nat (‹·›)
  | Imp form form (infixr ‹→› 100)

primrec semantics (infix ‹⊨› 50) where
  ‹I ⊨ ·n = I n› |
  ‹I ⊨ p → q = (I ⊨ p ⟶ I ⊨ q)›

abbreviation sc (‹⟦_⟧›) where ‹⟦I⟧ X Y ≡ (∀p ∈ set X. I ⊨ p) ⟶ (∃q ∈
set Y. I ⊨ q)›

inductive SC (infix ‹≫› 50) where
  Imp_L: ‹p → q # X ≫ Y› if ‹X ≫ p # Y› and ‹q # X ≫ Y› |
  Imp_R: ‹X ≫ p → q # Y› if ‹p # X ≫ q # Y› |
  Set_L: ‹X' ≫ Y› if ‹X ≫ Y› and ‹set X' = set X› |
  Set_R: ‹X ≫ Y'› if ‹X ≫ Y› and ‹set Y' = set Y› |
  Basic: ‹p # _ ≫ p # _›

function mp where
  ‹mp A B [] [] = (set A ∩ set B ≠ {})› |
  ‹mp A B ((p → q) # C) [] = (mp A B C [p] ∧ mp A B (q # C) [])› |
  ‹mp A B C ((p → q) # D) = mp A B (p # C) (q # D)› |
  ‹mp A B (·n # C) [] = mp (n # A) B C []› |
  ‹mp A B C (·n # D) = mp A (n # B) C D›
  by pat_completeness simp_all

termination
  by (relation ‹measure (λ(_, _, C, D). 2 * (∑p ← C @ D. size p) +
size (C @ D))›) simp_all

lemma main: ‹(∀I. ⟦I⟧ (map · A @ C) (map · B @ D)) ⟷ mp A B C D›
  by (induct rule: mp.induct) (auto 5 2)

definition prover (‹⊢›) where ‹⊢ p ≡ mp [] [] [] [p]›

theorem prover_correct: ‹⊢ p ⟷ (∀I. I ⊨ p)›
  unfolding prover_def by (simp flip: main)

export_code ⊢ in SML

lemma MP: ‹mp A B C D ⟹ set X ⊇ set (map · A @ C) ⟹ set Y ⊇ set
(map · B @ D) ⟹ X ≫ Y›
proof (induct A B C D arbitrary: X Y rule: mp.induct)
  case (1 A B)
  obtain n where ‹n ∈ set A› ‹n ∈ set B›
    using 1(1) by auto
  then have ‹set (·n # X) = set X› ‹set (·n # Y) = set Y›
    using 1(2,3) by auto
  then show ?case
    using Set_L Set_R Basic by metis
next
```

```
  case (2 A B p q C)
  have ‹set (map · A @ C) ⊆ set X› ‹set (map · B) ⊆ set (p # Y)›
    using 2(4,5) by auto
  moreover have ‹set (map · A @ C) ⊆ set (q # X)› ‹set (map · B) ⊆
set Y›
    using 2(4,5) by auto
  ultimately have ‹(p → q) # X ≫ Y›
    using 2(1-3) Imp_L by simp
  then show ?case
    using 2(4) Set_L by fastforce
next
  case (3 A B C p q D)
  have ‹set (map · A @ C) ⊆ set (p # X)› ‹set (map · B @ D) ⊆ set (q
# Y)›
    using 3(3,4) by auto
  then have ‹X ≫ (p → q) # Y›
    using 3(1,2) Imp_R by simp
  then show ?case
    using 3(4) Set_R by fastforce
qed simp_all

theorem OK: ‹(∀I. ⟦I⟧ X Y) ⟷ X ≫ Y›
  by (rule, use MP main[of ‹[]› _ ‹[]› _] in simp, induct rule:
SC.induct) auto

corollary ‹[] ≫ [p] ⟷ (∀I. I ⊨ p)›
  using OK by force

proposition ‹[] ≫ [p → p]›
proof -
  from Imp_R have ?thesis if ‹[p] ≫ [p]›
    using that by force
  with Basic show ?thesis
    by force
qed

proposition ‹[] ≫ [p → (p → q) → q]›
proof -
  from Imp_R have ?thesis if ‹[p] ≫ [(p → q) → q]›
    using that by force
  with Imp_R have ?thesis if ‹[p → q, p] ≫ [q]›
    using that by force
  with Imp_L have ?thesis if ‹[p] ≫ [p, q]› and ‹[q, p] ≫ [q]›
    using that by force
  with Basic show ?thesis
    by force
qed

proposition ‹[] ≫ [p → q → q → p]›
proof -
  from Imp_R have ?thesis if ‹[p] ≫ [q → q → p]›
    using that by force
  with Imp_R have ?thesis if ‹[q, p] ≫ [q → p]›
```

```
    using that by force
  with Imp_R have ?thesis if ‹[q, q, p] ⫸ [p]›
    using that by force
  with Set_L have ?thesis if ‹[p, q] ⫸ [p]›
    using that by force
  with Basic show ?thesis
    by force
qed

end
```