```
theory Fold_Assn
  imports Base
begin

definition fold_assn :: "assn list ⇒ assn" where
  "fold_assn assns = foldr (*) assns emp"

lemma fold_assn_emp [simp]: "fold_assn [] = emp"
  unfolding fold_assn_def by simp

lemma fold_assn_cons [simp]: "fold_assn (x#xs) = x * fold_assn xs"
  unfolding fold_assn_def by simp_all

lemma fold_assn_app [simp]: "fold_assn (xs@ys) = fold_assn xs * fold_assn ys"
  by(induction xs)(auto simp: algebra_simps)

lemma fold_assn_remove1: "x ∈L xs ⟹ fold_assn xs = x * fold_assn (remove1 x xs)"
  by(induction xs)(auto simp: algebra_simps)

lemma fold_assn_false [simp]: "false ∈L xs ⟹ fold_assn xs = false"
  using fold_assn_remove1 by auto

lemma fold_assn_emp_remove1 [simp]: "fold_assn (remove1 emp xs) = fold_assn xs"
  by(induction xs) auto

lemma fold_assn_emp_removeAll [simp]: "fold_assn (removeAll emp xs) = fold_assn xs"
  by(induction xs) auto

lemma fold_assn_remove1_map: "x ∈L xs
   ⟹ fold_assn (remove1 (f x) (map f xs)) = fold_assn (map f (remove1 x xs))"
proof(induction xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons a xs)
  then show ?case
    using fold_assn_remove1[of "f a" "map f xs"] image_iff
    by fastforce
qed

end
```