```
theory Master_Assn
  imports Cell Fold_Assn
begin

definition master_assn :: "('a cell ref * 'a::heap cell') list ⇒ assn" where
  "master_assn t = fold_assn (map (λ(p, c'). ∃A c. p ↦r c * cell_assn c' c) t)"

lemma open_master_assn_cons:
    "master_assn ((p, c') # t) = (∃A c. p ↦r c * cell_assn c' c) * master_assn t"

lemma open_master_assn':
  assumes "(p, c') ∈L t"
  shows "master_assn t =
           (∃A c. p ↦r c * cell_assn c' c) * master_assn (remove1 (p, c') t)"

lemma open_master_assn:
  assumes "(p, c') ∈L t"
  shows "master_assn t
            ⟹A (∃A c. p ↦r c * cell_assn c' c) * master_assn (remove1 (p, c') t)"

lemma close_master_assn_array: "(a, Array' xs) ∈L t
  ⟹ a' ↦a xs * a ↦r cell.Array a' * master_assn (remove1 (a, Array' xs) t)
       ⟹A master_assn t"

lemma close_master_assn_upd: "(a, Upd' i x a') ∈L t
     ⟹ a ↦r Upd i x a' * master_assn (remove1 (a, Upd' i x a') t) ⟹A master_assn t"

lemma close_master_assn_upd': "(a, Upd' i x a') ∈L t
     ⟹ a ↦r Upd i x a' * master_assn (remove1 (a, Upd' i x a') t) = master_assn t"

lemma master_assn_distinct: "h ⊨ master_assn t ⟹ distinct (map fst t)"

lemma master_assn_distinct': "master_assn t ⟹A master_assn t * ↑(distinct (map fst t))"

end
```