```
lemma hnr_frame:
  assumes
    "hnr Γ fi Γ' f"
    "Γ_P ⟹_A Γ * F"
  shows
    "hnr Γ_P fi (λr ri. Γ' r ri * F) f"

attribute_setup framed =
    ‹Scan.succeed (Thm.rule_attribute [] (fn _ => fn thm => @{thm hnr_frame} OF [thm, asm_
    ‹Add frame to hnr rule›

lemma frame_prepare:
  assumes
    "emp * P * emp ⟹_A emp * Q * F"
  shows
    "P ⟹_A Q * F"

lemma split_id_assn: "id_assn p pi = id_assn (fst p) (fst pi) * id_assn (snd p) (snd pi)"

method frame_norm_assoc = (simp only: mult.left_assoc[where 'a=assn] split_id_assn)?

method frame_prepare = rule frame_prepare, frame_norm_assoc

lemma frame_no_match:
  assumes
    "Ps1 * (P * Ps2) ⟹_A Qs * Q * F"
  shows
    "Ps1 * P * Ps2 ⟹_A Qs * Q * F"

lemma frame_match_pure:
  assumes
    "Ps1 * ↑(P) * Ps2 ⟹_A Qs * F"
  shows
    "Ps1 * ↑(P) * Ps2 ⟹_A Qs * ↑(P) * F"

lemma frame_match:
  assumes
    "P ⟹_A Q"
    "Ps1 * Ps2 ⟹_A Qs * F"
  shows
    "Ps1 * P * Ps2 ⟹_A Qs * Q * F"

lemma frame_match_emp:
   assumes
    "Ps ⟹_A Qs * F"
  shows
    "Ps ⟹_A Qs * emp * F"

lemma frame_done: "F * emp ⟹_A emp * F"

method frame_try_match methods match_atom = then_else
  ‹rule frame_match_pure | rule frame_match, (match_atom; fail) | rule frame_match_emp›
  ‹frame_norm_assoc›
  ‹rule frame_no_match, frame_try_match match_atom›

method frame_done = simp only: assn_one_left mult_1_right[where 'a=assn], rule ent_refl

method hnr_frame_inference methods match_atom =
  frame_prepare, (frame_try_match match_atom)+, frame_done
```