```
theory Find imports Post.nomad "HSBCh_pre_nomad/Find_state" begin

(* Find ------------------------------------------------------------------- *)


definition find_step :: "
('k,'v,'frame,'node) frame_ops =>
('v,('node,(nat)option,'v) store_ops =>
('k,('node,(nat)option,'v)find_state,'v,'leaf,'frame) find_state,'v) MP' where
'find_step frame_ops store_ops s =
  let read = store_ops(read in
  s >>=
  case fs of
    F_finished _ => (failwith STR'' ''find_step'') |
    F_start k => (
      read r (lmap (k,1) .
      return (
      find_node s r) |
      find_node s r =
        let frm = (frame_ops(split_node_at_key) r k in
        let r' = (frame_ops(midpoint) frm in
        if_then(r'!=r)(read r') .
        (_find_leaf leaf r) F_finished(k,v,s,(leaf_sel)))")


definition find_big_step :: "
('k,'v,'frame,'node) frame_ops =>
('v,('node,(nat)option,'v) store_ops =>
('k,('node,(nat)option,'v)find_state,'v,'leaf,'frame) find_state,'v) MP' where
'find_big_step frame_ops store_ops i =
  let step = find_step frame_ops store_ops in
  i >>=
  fix s.
  case s of
    F_finished _ => (return Nomo) |
    _ => (step s >> (Keep Nomo))
  )")


definition find :: "
('k,'v,'frame,'node) frame_ops =>
('v,('node,(nat)option,'v) store_ops =>
('k,'v,'k,'v) f0 => ('leaf * ('frame list,'v)) MP' where
'find = (
  find_big_step frame_ops store_ops s .
  find_big_step frame_ops store_ops s >> kind (k =.
  case v of
  F_finished(k',v',leaf,sel) => return (v,leaf,sel)
  | _ => failwith STR'' ''find 2'')")


(* attempt to do the same, but within a locale *)

locale f =
  fixes ops :: 'constants' and
  fixes k :: 'k' and
  frame_ops :: ('k,'v,'frame,'left_half,'right_half,'node) frame_ops' and
  store_ops :: ('v,('node,'leafinfo,'v) store_ops'


definition (in f) find_step :: "
('k,'v,'frame,'node) find_state => ('k,'v,'leaf,'frame) find_state,'v) MP' where
'find_step' =
  let read = store_ops(read ops
  s >>=
  case fs of
    F_finished _ => (failwith STR'' ''find_step 2'') |
    F_start k => (
      read r (lmap (k,1) .
      return (
      case r of
      Node(_,_,_,_) => (
        let frm = (frame_ops(split_node_at_key) r k in
        let r' = (frame_ops(midpoint) frm in
        if_then(r'!=r)(read r') .
        (_find_leaf leaf r) F_finished(k,v,leaf,sel))")

definition (in f) find_big_step :: "
('k,'v,'frame,'node) find_state => ('k,'v,'leaf,'frame) find_state,'v) MP' where
'find_big_step =
  fix s.
  case s of
    F_finished _ => (return Nomo) |
    _ => (find_step s >> (Keep Nomo))
  )")


definition (in f) find :: "f' => 'k => ('v * 'leaf * ('frame list,'v) MP' where
'find = (
  find_big_step s (initial_find_state k = in
  find_big_step s >> kind (k =.
  case v of
  F_finished(k',v',leaf,sel) => return (v,leaf,sel)
  | _ => failwith STR'' ''find 3'')")

print_locale' f

end f.find_def'


definition find0 :: "
('k,'v,'frame,'left_half,'right_half,'node) frame_ops =>
('v,('node,(nat)option,'v) store_ops =>
('k,'v,'k,'v) f0 => ('leaf * ('frame list,'v) MP' where
'find0 = (

lemma find_def_0: "find0 p q r k = undefined"
  apply(simp add: find0_def)
  apply(simp add: find_state.case_eosp add: f.find_def f.find_big_step_def f.find_step_def)
  done

lemma find_def_find0: "find0 = (\<lambda> p \<lambda> q \<lambda> r \<lambda> k =
(let s = (make_initial_find_state k ;
    v = (case (find_step p q r s) of
      (F_start s') =>
        (case (find_step p q r s') of
          (F_finished (k',v',leaf,sel) =>
            (k',v',leaf,sel)
          | _ =>
            (Temp_state
              (\<r. (let frm = (r (r (r split_node_at_key) s) k in (r (r midpoint) frm, frm.# s)))
              \<lambda> F_finished (k, b, c, leaf, sel)) |
          Temp_state
            \<r. F_finished s \<> return Nomo) |>
        Keep_state s) |>
      (temp_option (return s)
        (\<s.\<
          (case (find_state
            \<.(case
              (case proof s\<
                \<s. (s (name s)
                \<s. (s (name) = \<>
                Temp
                  (Temp_state
                    (\<r. (let frm = (r (r (r split_node_at_key) s) k in (r (r midpoint) frm, frm.# s)))
                    \<lambda> F_finished (k, b, c, leaf, sel)) |>
                Temp Nomo)
                \<. return Nomo)|) |>
            Send (\<s. (s (name proof \<.failwith STR''find 5'' | F_finished (k, b, bar) \<.return bar)"
  apply(simp add: find0_def)
  apply(simp add: find_state.case_eosp add: f.find_def f.find_big_step_def f.find_step_def)
  done
"}

end


(* old ------------------------------------------------------------------- *)


(* find_trans ------------------------------------------------------------- *)

(*
definition find_trans :: "(trace * 'fs) trans_f' where
"find_trans = (\<.fs.(\<.(k'.(k'=(k) | (\<lambda>(k,v).(k'=k)) (f0 fs)))"
*)

(* Lemma ------------------------------------------------------------------ *)

(* wf_fts is invariant ---------------------------------------------------- *)

definition invariant_lem :: "bool" where
"invariant_lem = (
  (\<.fs. (\<. (_,fs) = _ \<.fs in wellformed_find_state s \<> fs) = f0) --- invariant find_trans f0)"
*)

*)
```