```
theory Insert imports Find "$SRC/b_pre_monad/Insert_state" begin

type_synonym ('k,'v,'r) fo = "('k,'v,'r)i12_t"
type_synonym ('k,'v,'r,'leaf,'frame) d (* down_state *) = "('k,'r,'leaf,'frame)find_state*'v"
type_synonym ('k,'v,'r,'frame) u (* up_state *) = "('k,'v,'r)fo*'frame list"

(* insert ------------------------------------------------------------ *)


definition step_down :: "
constants ⇒
'k ord ⇒
('k,'r,'frame,'left_half,'right_half,'node) frame_ops ⇒
('r,('node,'leaf)dnode,'t) store_ops ⇒
('k,'v,'r,'leaf,'frame) d ⇒ (('k,'v,'r,'leaf,'frame) d,'t) MM" where
"step_down cs k_cmp frame_ops store_ops = (
  let find_step =  find_step cs k_cmp frame_ops store_ops in
  (% d.
  let (fs,v) = d in
  find_step fs |> fmap (% d'. (d',v)) ))"


definition step_bottom :: "
constants ⇒
'k ord ⇒
('k,'v,'leaf) leaf_ops ⇒
('k,'r,'node) node_ops ⇒
('r,('node,'leaf)dnode,'t) store_ops ⇒
('k,'v,'r,'leaf,'frame) d ⇒ (('k,'v,'r,'frame) u + unit,'t) MM" where
"step_bottom cs k_cmp leaf_ops node_ops store_ops d = (
  let (write,rewrite) = (store_ops|>wrte,store_ops|>rewrite) in
  let mk_leaf = (leaf_ops|>mk_leaf) in
  let (fs,v) = d in
  case dest_F_finished fs of
  None ⇒ (failwith (STR ''insert, step_bottom, 1''))
  | Some(r0,k,r,leaf,stk) ⇒ (
    — ‹ free here? FIXME ›
    let leaf' = (leaf_ops|>leaf_insert) k v leaf in
    case (leaf_ops|>leaf_length) leaf' ≤ cs|>max_leaf_size of
    True ⇒ (
      — ‹ we want to update in place if possible ›
      Disk_leaf leaf' |> rewrite r |> bind (% r'.
      case r' of
      None ⇒
        — ‹ block was updated in place ›
        return (Inr ())
      | Some r' ⇒ return (Inl(I1 r',stk))))
    | False ⇒ (
      let kvs' = (leaf_ops|>leaf_kvs) leaf' in
      let (kvs1,k',kvs2) = split_leaf cs kvs' in
      Disk_leaf (mk_leaf kvs1) |> write |> bind (% r1.
      Disk_leaf (mk_leaf kvs2) |> write |> bind (% r2.
      return (Inl(I2(r1,k',r2),stk)))))))"



definition step_up :: "
constants ⇒
'k ord ⇒
('k,'r,'node) node_ops ⇒
('k,'r,'frame,'left_half,'right_half,'node) frame_ops ⇒
('r,('node,'leaf)dnode,'t) store_ops ⇒
('k,'v,'r,'frame) u ⇒ (('k,'v,'r,'frame) u + unit,'t) MM" where
"step_up  cs k_cmp node_ops frame_ops store_ops u = (
  let (write,rewrite) = (store_ops|>wrte,store_ops|>rewrite) in
  let (fo,stk) = u in
  case stk of
  [] ⇒ failwith (STR ''insert, step_up,1'')
  | frm#stk' ⇒ (
    let (lh,rh) = ((frame_ops|>left_half) frm, (frame_ops|>right_half) frm) in
    let original_r = (frame_ops|>original_node_r) frm in
    case fo of
    I1 r ⇒ (
      let n = (frame_ops|>unsplit) (lh,R(r),rh) in
      Disk_node(n) |> rewrite original_r |> bind (% r2.
      case r2 of
      None ⇒ return (Inr ())
      | Some r2 ⇒ return (Inl (I1 r2, stk'))))
    | I2 (r1,k,r2) ⇒ (
      let n = (frame_ops|>unsplit) (lh, Rkr(r1,k,r2), rh) in
      let n = (n :: 'node) in
      case (node_ops|>node_keys_length) n ≤ (cs|>max_node_keys) of
      True ⇒ (
        Disk_node(n) |> rewrite original_r |> bind (% r2.
        case r2 of
        None ⇒ return (Inr ())
        | Some r2 ⇒ return (Inl (I1 r2, stk'))))
      | False ⇒ (
        let (n1,k,n2) = (node_ops|>split_large_node) n in
        Disk_node(n1) |> write |> bind (% r1.
        Disk_node(n2) |> write |> bind (% r2.
        return (Inl (I2(r1,k,r2),stk'))))) )))"



definition insert_step :: "
constants ⇒
'k ord ⇒
('k,'v,'leaf) leaf_ops ⇒
('k,'r,'node) node_ops ⇒
('k,'r,'frame,'left_half,'right_half,'node) frame_ops ⇒
('r,('node,'leaf)dnode,'t) store_ops ⇒
('k,'v,'r,'leaf,'frame) insert_state ⇒ (('k,'v,'r,'leaf,'frame) insert_state,'t) MM" where
"insert_step cs k_cmp leaf_ops node_ops frame_ops store_ops = (
  let step_down = step_down cs k_cmp frame_ops store_ops in
  let step_bottom = step_bottom cs k_cmp leaf_ops node_ops store_ops in
  let step_up = step_up  cs k_cmp node_ops frame_ops store_ops in
  let write = store_ops|>wrte in
  (% s.
  case s of
  I_down d ⇒ (
    let (fs,v) = d in
    case dest_F_finished fs of
    None ⇒ (step_down d |> fmap (% d. I_down d))
    | Some _ ⇒ step_bottom d |> bind (% bot.
      case bot of
      Inr () ⇒ return I_finished_with_mutate
      | Inl u ⇒ return (I_up u)))
  | I_up u ⇒ (
    let (fo,stk) = u in
    case stk of
    [] ⇒ (
      case fo of
      I1 r ⇒ return (I_finished r)
      | I2(r1,k,r2) ⇒ (
        (Disk_node((node_ops|>node_make_small_root)(r1,k,r2)) |> write |> bind (% r.
        return (I_finished r)))))
    | _ ⇒ (step_up u |> bind (% u.
      case u of
      Inr () ⇒ return I_finished_with_mutate
      | Inl u ⇒ return (I_up u))))
  | I_finished _ ⇒ (failwith (STR ''insert_step 1''))
  | I_finished_with_mutate ⇒ (failwith (STR ''insert_step 2''))))"

definition insert_big_step :: "
constants ⇒
'k ord ⇒
('k,'v,'leaf) leaf_ops ⇒
('k,'r,'node) node_ops ⇒
('k,'r,'frame,'left_half,'right_half,'node) frame_ops ⇒
('r,('node,'leaf)dnode,'t) store_ops ⇒
('k,'v,'r,'leaf,'frame) insert_state ⇒ (('k,'v,'r,'leaf,'frame) insert_state,'t) MM" where
"insert_big_step cs k_cmp leaf_ops node_ops frame_ops store_ops = (
  let insert_step = insert_step cs k_cmp leaf_ops node_ops frame_ops store_ops in
  (% i.
  iter_m (% i. case i of
    I_finished r ⇒ (return None)
    | I_finished_with_mutate ⇒ (return None)
    | _ ⇒ (insert_step i |> fmap Some))
    i))"

definition insert :: "
constants ⇒
'k ord ⇒
('k,'v,'leaf) leaf_ops ⇒
('k,'r,'node) node_ops ⇒
('node ⇒ 'r s) ⇒
('node ⇒ 'k s) ⇒
('k,'r,'frame,'left_half,'right_half,'node) frame_ops ⇒
('r,('node,'leaf)dnode,'t) store_ops ⇒
'r ⇒ 'k ⇒ 'v ⇒ ('r option,'t) MM" where
"insert cs k_cmp leaf_ops node_ops node2rs node2ks frame_ops store_ops r k v = (
  let check_tree_at_r = check_tree_at_r cs k_cmp leaf_ops node_ops node2rs node2ks store_ops in
  let i = make_initial_insert_state r k v in
  insert_big_step cs k_cmp leaf_ops node_ops frame_ops store_ops i |> bind (% i.
  case i of
  I_finished r ⇒ (check_tree_at_r r |> bind (% _. return (Some r)))
  | I_finished_with_mutate ⇒ (check_tree_at_r r |> bind (% _. return None))
  | _ ⇒ failwith (STR ''insert 1'')
))"



end





(*
export_code
"Code_Numeral.int_of_integer"
fmap
Disk_node
make_constants
make_store_ops
make_initial_find_state
I1
I_down
insert_step

in OCaml file "/tmp/insert_with_mutation.ml"
*)
```