Module `Isa_btree.Isa_export_wrapper`

Wrap Isabelle-exported code in an OCaml-friendly interface

Isabelle test flag

Control isabelle assert flag

```
val enable_isa_checks : unit -> unit
```

```
val disable_isa_checks : unit -> unit
```

Misc

```
val dest_Some : 'a option -> 'a
```

```
type ('node, 'leaf) dnode = ('node, 'leaf) Isa_export.Disk_node.dnode
```

Recall `dnode` type

Pre-map operations

```
type ('k, 'v, 'r, 'leaf, 'frame, 't) pre_map_ops = {
    find : r:'r -> k:'k -> ('r * 'leaf * 'frame list, 't)
    Tjr_monad.Types.m;
    insert : r:'r -> k:'k -> v:'v -> ('r option, 't) Tjr_monad.Types.m;
    delete : r:'r -> k:'k -> ('r, 't) Tjr_monad.Types.m;
}
```

Pre-map ops, with an explicit root pointer

Leaf operations

```
type ('k, 'v, 'leaf) leaf_ops = {
    leaf_lookup : 'k -> 'leaf -> 'v option;
    leaf_insert : 'k -> 'v -> 'leaf -> 'leaf * 'v option;
    leaf_remove : 'k -> 'leaf -> 'leaf;
    leaf_length : 'leaf -> int;
    dbg_leaf_kvs : 'leaf -> ('k * 'v) list;
    leaf_steal_right : ('leaf * 'leaf) -> 'leaf * 'k * 'leaf;
    leaf_steal_left : ('leaf * 'leaf) -> 'leaf * 'k * 'leaf;
    leaf_merge : ('leaf * 'leaf) -> 'leaf;
    split_large_leaf : int -> 'leaf -> 'leaf * 'k * 'leaf;
}
```

As Isabelle def. See \doc(doc:leaf_ops).

```
module Internal_leaf_impl : sig ... end
```

```
type ('k, 'v) _leaf_impl = ('k, 'v, unit) Tjr_poly_map.map
```

```
val make_leaf_ops : k_cmp:('k -> 'k -> int) -> ('k, 'v, ('k, 'v) _leaf_impl)
leaf_ops
```

Node operations

```
type ('k, 'r, 'node) node_ops = {
    split_node_at_k_index : int -> 'node -> 'node * 'k * 'node;
    node_merge : ('node * 'k * 'node) -> 'node;
    node_steal_right : ('node * 'k * 'node) -> 'node * 'k * 'node;
    node_steal_left : ('node * 'k * 'node) -> 'node * 'k * 'node;
    node_keys_length : 'node -> int;
    node_make_small_root : ('r * 'k * 'r) -> 'node;
    node_get_single_r : 'node -> 'r;
    check_node : 'node -> unit;
    dbg_node_krs : 'node -> 'k list * 'r list;
}
```

```
module Internal_node_impl : sig ... end
```

```
type ('k, 'r) _node_impl = ('k option, 'r, unit) Tjr_poly_map.map
```

```
val make_node_ops : k_cmp:('k -> 'k -> int) -> ('k, 'r, ('k, 'r) _node_impl)
node_ops
```

Frame operations

```
module Internal_bottom_or_top : sig ... end
```

```
type ('k, 'r) segment = 'k Internal_bottom_or_top.or_bottom * 'r * ('k * 'r)
list * 'k Internal_bottom_or_top.or_top
```

```
type ('k, 'r, 'frame, 'node) frame_ops = {
    split_node_on_key : 'r -> 'k -> 'node -> 'frame;
    midpoint : 'frame -> 'r;
    get_focus : 'frame -> 'k Internal_bottom_or_top.or_bottom * 'r * 'k
    Internal_bottom_or_top.or_top;
    get_focus_and_right_sibling : 'frame -> ('k
    Internal_bottom_or_top.or_bottom * 'r * 'k * 'r * 'k
    Internal_bottom_or_top.or_top) option;
    get_left_sibling_and_focus : 'frame -> ('k
    Internal_bottom_or_top.or_bottom * 'r * 'k * 'r * 'k
    Internal_bottom_or_top.or_top) option;
    replace : ('k, 'r) segment -> ('k, 'r) segment -> 'frame -> 'frame;
    frame_to_node : 'frame -> 'node;
    get_midpoint_bounds : 'frame -> 'k option * 'k option;
    backing_node_blk_ref : 'frame -> 'r;
    dbg_frame : 'frame -> unit;
}
```

See Isabelle defn. See \doc(doc:frame_ops)

```
module Internal_frame_impl : sig ... end
```

```
type ('k, 'r) _frame_impl = ('k, 'r, ('k, 'r) _node_impl)
Internal_frame_impl.frame
```

```
val make_frame_ops : k_cmp:('a -> 'a -> int) -> dbg_frame:(('a, 'b)
_frame_impl -> unit) -> ('a, 'b, ('a, 'b) _frame_impl, ('a, 'b) _node_impl)
frame_ops
```

Store operations

```
type ('r, 'dnode, 't) store_ops = {
    read : 'r -> ('dnode, 't) Tjr_monad.Types.m;
    wrte : 'dnode -> ('r, 't) Tjr_monad.Types.m;
    rewrite : 'r -> 'dnode -> ('r option, 't) Tjr_monad.Types.m;
    free : 'r list -> (unit, 't) Tjr_monad.Types.m;
}
```

Conversions between Isabelle types and OCaml types

```
module Internal_conversions : sig ... end
```

```
module Internal_make_find_insert_delete : sig ... end
```

Recap, packaging and export

```
module Internal_export : sig ... end
```

```
include Internal_export
```

```
type ('k, 'r) node_impl
```

```
type ('k, 'v) leaf_impl
```

```
type ('k, 'r) frame_impl
```

```
val make_find_insert_delete : monad_ops:'a Tjr_monad.Types.monad_ops ->
cs:Constants_type.constants -> k_cmp:('k -> 'k -> int) -> store_ops:
('r, (('k, 'r) node_impl, ('k, 'v) leaf_impl) dnode, 'a) store_ops ->
check_tree_at_r':('r -> (bool, 'a) Tjr_monad.Types.m) -> dbg_frame:(('k, 'r)
frame_impl -> unit) -> ('k, 'v, 'r, ('k, 'v) leaf_impl, ('k, 'r)
frame_impl, 'a) pre_map_ops
```

```
val wf_tree : cs:Constants_type.constants -> ms:Isa_export.Tree.min_size_t
option -> k_cmp:('a -> 'a -> int) -> ('a, 'b) Isa_export.Tree.tree -> bool
```