```
type num

module Linear = struct
  type linexpr = num array
  type affexpr = {
    lin : linexpr;
    cst : num;
  }
  type typcond = Sup | SupEg
  type cond = typcond * affexpr

  type generator
    | Point of affexpr
    | Ray of linexpr
    | Line of linexpr
end

type bound = num option
type interval = {
  inf : bound
  sup : bound
}


module type A = sig
  type t
    (** The type of abstract values *)

  (** *)
  val minimize : t -> unit
  val canonicalize : t -> unit

  (** Constructors and basic operations *)
  val bottom : int -> t
  val top : int -> t
  val meet : t -> t -> t
  val join : t -> t -> t
  val meet_linearcond : t -> Linear.cond -> t
  val meet_nonlinearcond : t -> NonLinear.cond -> t

  (** Tests *)
  val is_bottom : t -> bool
  val is_top : t -> bool
  val is_leq : t -> t -> bool
  val is_eq : t -> t -> bool
  val is_leq_linearcond : t -> Linear.cond -> bool

  (** Accessors *)
  val dim : t -> int
  val to_constraints : t -> Linear.cond list
  val to_generators : t -> Linear.generator list
  val to_bounds : t -> interval array
  val bound_linear_expr : t -> Linear.affexpr -> interval

  (** Widening *)
  val widening : t -> t -> t
    (** Standard widening *)
```

```
  val limited_widening : t -> t -> Linear.cond list -> t
    (** Widening with thresholds *)

  (** Projections, Embeddings, ... *)
  (* At the end *)
  val add_dims_and_embed : t -> int -> t
  val add_dims_and_project : t -> int -> t
  val del_dims : t -> int -> t
  (* Anywhere *)
  val add_permute_dims_and_embed : t -> int -> int array -> t
  val add_permute_dims_and_project : t -> int -> int array -> t
  val permute_del_dims : t -> int -> int array -> t

  (** Transformations *)
  val assign_var_linear : t -> int -> Linear.affexpr -> t
  val substitute_var_linear : t -> int -> Linear.affexpr -> t
  val assign_vars_linear : t -> (int * Linear.affexpr) array -> t
  val substitute_vars_linear : t -> (int * Linear.affexpr) array -> t

end
```