```
qualified partial_function (heap) realize ::
  "('a::heap) diff_arr ⇒ 'a array Heap"
where
  "realize diff_arr = do {
    cell ← !diff_arr;
     case cell of
        Array arr ⇒ do {
            len ← Array.len arr;
            xs  ← Array.freeze arr;
            Array.make len (List.nth xs)
        }
      | Upd i v diff_arr ⇒ do {
          arr ← realize diff_arr;
          Array.upd i v arr
        }
  }"

qualified partial_function (heap) update ::
    "('a::heap) diff_arr ⇒ nat ⇒ 'a::heap ⇒ 'a diff_arr Heap"
where
  "update diff_arr i v = do {
      cell ← !diff_arr;
      case cell of
        Array arr ⇒ do {
          new_diff_arr ← ref (Array arr);
          old_v ← Array.nth arr i;
          diff_arr :=R Upd i old_v new_diff_arr;
          Array.upd i v arr;
          return new_diff_arr
        }
      | Upd _ _ _ ⇒ do {
          arr ← realize diff_arr;
          Array.upd i v arr;
          ref (Array arr)
        }
  }"

lemma realize [sep_heap_rules]:
  "<master_assn t * ↑(t ⊢ xs ∼ diff_arr)>
     realize diff_arr
   <λarr. master_assn t * arr ↦a xs>"

lemma update_diff_arr_rel: "⟦
  i < List.length xs;
  (diff_arr, Array' xs) ∈L t;
  distinct (map fst t);
  t ⊢ xs' ∼⇩n' diff_arr'
⟧ ⟹ ∃n. (new_diff_arr, Array' (xs[i := v])) #
          (diff_arr, Upd' i (xs ! i) new_diff_arr) #
          remove1 (diff_arr, Array' xs) t ⊢ xs' ∼⇩n diff_arr'"

lemma update [sep_heap_rules]:
  "<master_assn t * ↑(t ⊢ xs ∼ diff_arr ∧ i < List.length xs)>
     update diff_arr i v
   <λdiff_arr. ∃At'. master_assn t' *
    ↑((∀xs' diff_arr'. t ⊢ xs' ∼ diff_arr' ⟶ t' ⊢ xs' ∼ diff_arr') ∧
      (t' ⊢ xs[i := v] ∼ diff_arr))>"
```