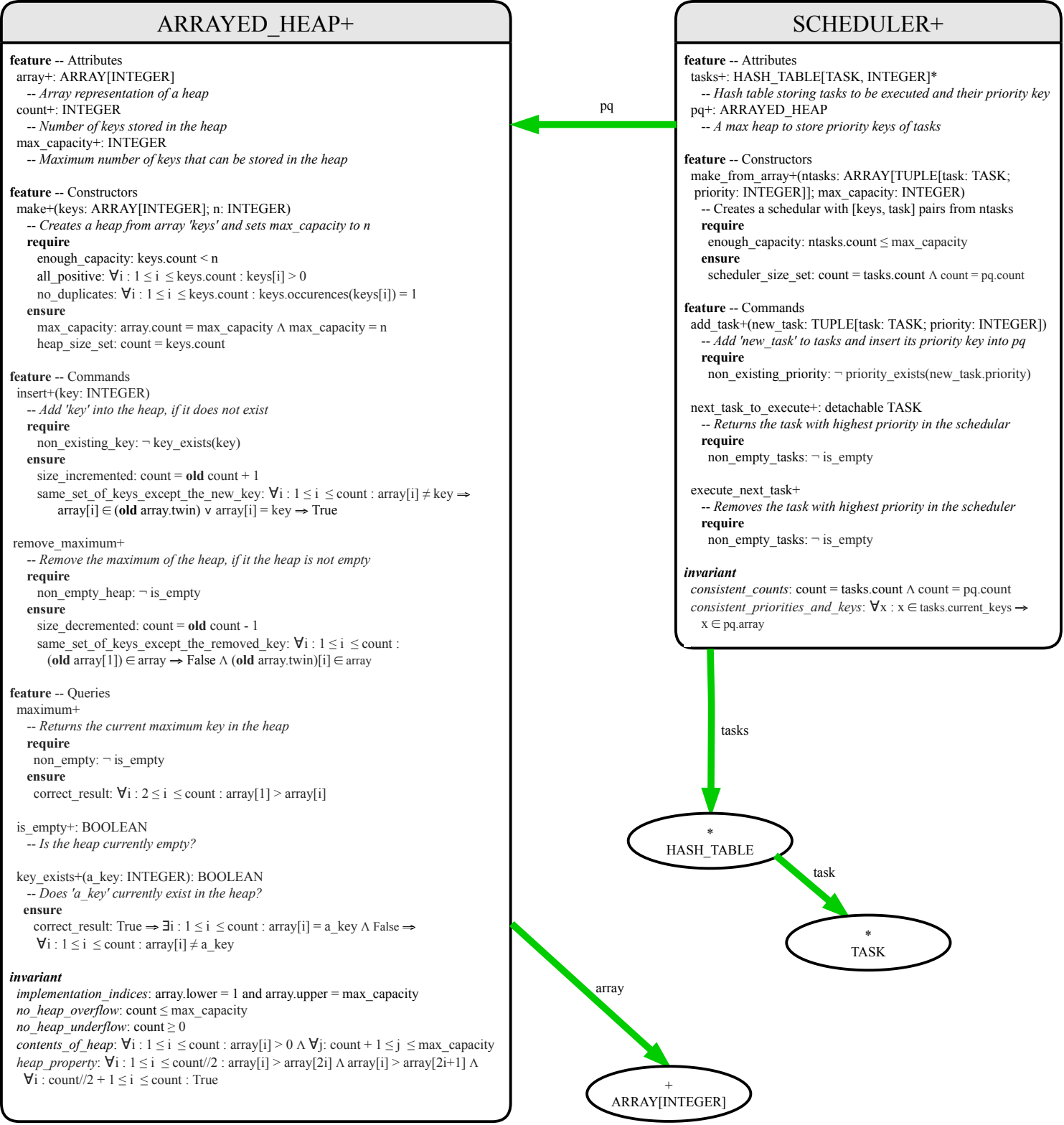
ARRAYED_HEAP+
feature -- Attributes
array+: ARRAY[INTEGER]
-- Array representation of a heap
count+: INTEGER
-- Number of keys stored in the heap
max_capacity+: INTEGER
-- Maximum number of keys that can be stored in the heap
feature -- Constructors
make+(keys: ARRAY[INTEGER]; n: INTEGER)
-- Creates a heap from array 'keys' and sets max_capacity to n
require
enough_capacity: keys.count < n
all_positive: ∀i : 1 ≤ i ≤ keys.count : keys[i] > 0
no_duplicates: ∀i : 1 ≤ i ≤ keys.count : keys.occurences(keys[i]) = 1
ensure
max_capacity: array.count = max_capacity ∧ max_capacity = n
heap_size_set: count = keys.count
feature -- Commands
insert+(key: INTEGER)
-- Add 'key' into the heap, if it does not exist
require
non_existing_key: ¬ key_exists(key)
ensure
size_incremented: count = old count + 1
same_set_of_keys_except_the_new_key: ∀i : 1 ≤ i ≤ count : array[i] ≠ key ⇒
array[i] ∈ (old array.twin) ∨ array[i] = key ⇒ True
remove_maximum+
-- Remove the maximum of the heap, if it the heap is not empty
require
non_empty_heap: ¬ is_empty
ensure
size_decremented: count = old count - 1
same_set_of_keys_except_the_removed_key: ∀i : 1 ≤ i ≤ count :
(old array[1]) ∈ array ⇒ False ∧ (old array.twin)[i] ∈ array
feature -- Queries
maximum+
-- Returns the current maximum key in the heap
require
non_empty: ¬ is_empty
ensure
correct_result: ∀i : 2 ≤ i ≤ count : array[1] > array[i]
is_empty+: BOOLEAN
-- Is the heap currently empty?
key_exists+(a_key: INTEGER): BOOLEAN
-- Does 'a_key' currently exist in the heap?
ensure
correct_result: True ⇒ ∃i : 1 ≤ i ≤ count : array[i] = a_key ∧ False ⇒
∀i : 1 ≤ i ≤ count : array[i] ≠ a_key
invariant
implementation_indices: array.lower = 1 and array.upper = max_capacity
no_heap_overflow: count ≤ max_capacity
no_heap_underflow: count ≥ 0
contents_of_heap: ∀i : 1 ≤ i ≤ count : array[i] > 0 ∧ ∀j: count + 1 ≤ j ≤ max_capacity
heap_property: ∀i : 1 ≤ i ≤ count//2 : array[i] > array[2i] ∧ array[i] > array[2i+1] ∧
∀i : count//2 + 1 ≤ i ≤ count : True
SCHEDULER+
feature -- Attributes
tasks+: HASH_TABLE[TASK, INTEGER]*
-- Hash table storing tasks to be executed and their priority key
pq+: ARRAYED_HEAP
-- A max heap to store priority keys of tasks
feature -- Constructors
make_from_array+(ntasks: ARRAY[TUPLE[task: TASK;
priority: INTEGER]]; max_capacity: INTEGER)
-- Creates a schedular with [keys, task] pairs from ntasks
require
enough_capacity: ntasks.count ≤ max_capacity
ensure
scheduler_size_set: count = tasks.count ∧ count = pq.count
feature -- Commands
add_task+(new_task: TUPLE[task: TASK; priority: INTEGER])
-- Add 'new_task' to tasks and insert its priority key into pq
require
non_existing_priority: ¬ priority_exists(new_task.priority)
next_task_to_execute+: detachable TASK
-- Returns the task with highest priority in the schedular
require
non_empty_tasks: ¬ is_empty
execute_next_task+
-- Removes the task with highest priority in the scheduler
require
non_empty_tasks: ¬ is_empty
invariant
consistent_counts: count = tasks.count ∧ count = pq.count
consistent_priorities_and_keys: ∀x : x ∈ tasks.current_keys ⇒
x ∈ pq.array
pq
tasks
*
HASH_TABLE
task
*
TASK
array
+
ARRAY[INTEGER]