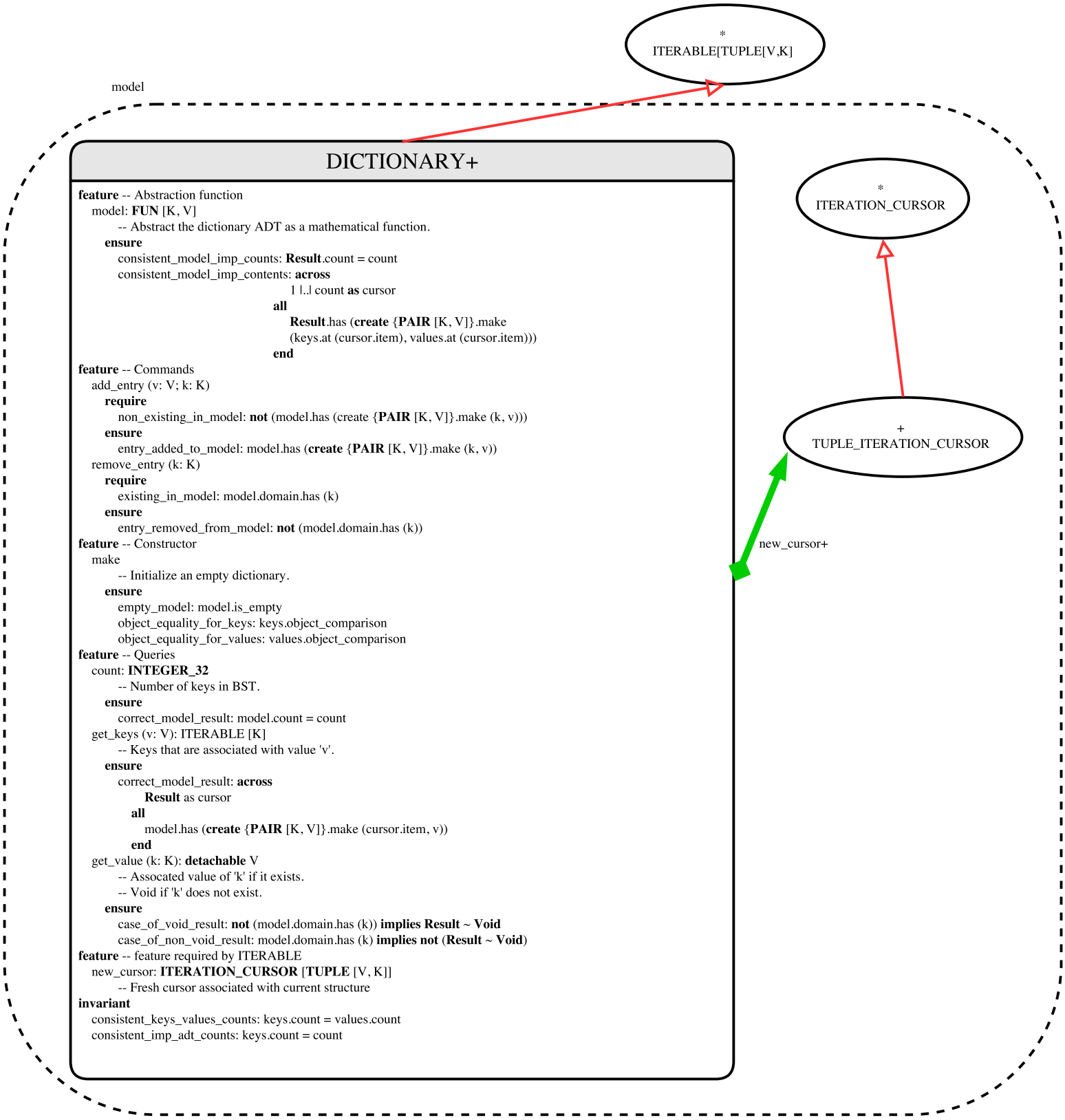

feature -- Abstraction function
  model: **FUN** [K, V]
    -- Abstract the dictionary ADT as a mathematical function.
  ensure
    consistent_model_imp_counts: **Result**.count = count
    consistent_model_imp_contents: **across**
                1 |..| count **as** cursor
              all
                Result.has (**create** {**PAIR** [K, V]}.make
                (keys.at (cursor.item), values.at (cursor.item)))
              end
feature -- Commands
  add_entry (v: V; k: K)
  require
    non_existing_in_model: **not** (model.has (create {**PAIR** [K, V]}.make (k, v)))
  ensure
    entry_added_to_model: model.has (**create** {**PAIR** [K, V]}.make (k, v))
  remove_entry (k: K)
  require
    existing_in_model: model.domain.has (k)
  ensure
    entry_removed_from_model: **not** (model.domain.has (k))
feature -- Constructor
  make
    -- Initialize an empty dictionary.
  ensure
    empty_model: model.is_empty
    object_equality_for_keys: keys.object_comparison
    object_equality_for_values: values.object_comparison
feature -- Queries
  count: **INTEGER_32**
    -- Number of keys in BST.
  ensure
    correct_model_result: model.count = count
  get_keys (v: V): ITERABLE [K]
    -- Keys that are associated with value 'v'.
  ensure
    correct_model_result: **across**
        Result as cursor
      all
        model.has (**create** {**PAIR** [K, V]}.make (cursor.item, v))
      end
  get_value (k: K): **detachable** V
    -- Assocated value of 'k' if it exists.
    -- Void if 'k' does not exist.
  ensure
    case_of_void_result: **not** (model.domain.has (k)) **implies Result** ~ **Void**
    case_of_non_void_result: model.domain.has (k) **implies not** (**Result** ~ **Void**)
feature -- feature required by ITERABLE
  new_cursor: **ITERATION_CURSOR** [**TUPLE** [V, K]]
    -- Fresh cursor associated with current structure
invariant
  consistent_keys_values_counts: keys.count = values.count
  consistent_imp_adt_counts: keys.count = count