The figures that follow are what should have appeared in the following paper, except for some problems that arose between the final page proofs and the printer:


Please consider these figures as replacements for the ones appearing in the published work.
concept Sorting_Machine_Template

context

  global context

  facility Standard_Boolean_Facility
  facility Standard_Integer_Facility

parametric context

  type Item

  math operation ARE_ORDERED (    
    x: math[Item] 
    y: math[Item] 
  ): boolean

  restriction  (* ARE_ORDERED is a total pre-ordering *)

local context

  math subtype INVENTORY_FUNCTION is
    function from math[Item] to integer
  exemplar  f
  constraint  for all x: math[Item] 
    (f(x) >= 0)

  math operation EMPTY_INVENTORY: 
    INVENTORY_FUNCTION
  definition  for all x: math[Item] 
    (EMPTY_INVENTORY (x) = 0)

  math operation IS_FIRST (    
    f: INVENTORY_FUNCTION 
    x: math[Item] 
  ): boolean
  definition  f(x) > 0 and
    for all y: math[Item] where
    ARE_ORDERED (y, x) and
    not ARE_ORDERED (x, y) 
    (f(y) = 0)
interface Sorting_Machine_State is modeled by
{
    count: INVENTORY_FUNCTION
    insertion_phase: boolean
}

exemplar m
initialization
ensures m = (EMPTY_INVENTORY, true)

operation Change_To_Insertion_Phase ( alters m: Sorting_Machine_State )
requires not m.insertion_phase
ensures m = (EMPTY_INVENTORY, true)

operation Insert ( alters m: Sorting_Machine_State consumes x: Item )
requires m.insertion_phase
ensures differ (m.count, #m.count, (#x)) and
m.count(#x) = #m.count(#x) + 1 and
m.insertion_phase

operation Change_To_Extraction_Phase ( alters m: Sorting_Machine_State )
requires m.insertion_phase
ensures m = (#m.count, false)

operation Extract ( alters m: Sorting_Machine_State produces x: Item )
requires m.count /= EMPTY_INVENTORY and not m.insertion_phase
ensures IS_FIRST (#m.count, x) and
differ (m.count, #m.count, (x)) and
m.count(x) = #m.count(x) - 1 and
not m.insertion_phase

operation Size ( preserves m: Sorting_Machine_State ): Integer
ensures Size = sum x: math[Item] (m.count(x))
operation Is_In_Insertion_Phase (  
    preserves m: Sorting_Machine_State  
): Boolean  

ensures Is_In_Insertion_Phase  iff  
    m.insertion_phase  

end Sorting_Machine_Template
concept Spanning_Forest_Machine_Template

context

global context

facility Standard_Boolean_Facility
facility Standard_Integer_Facility

parametric context

constant max_vertex: Integer

restriction max_vertex > 0

local context

math subtype EDGE is
  v1: integer
  v2: integer
  w: integer

exemplar e

constraint 1 <= e.v1 <= max_vertex and
           1 <= e.v2 <= max_vertex and
           e.w > 0

math subtype GRAPH is set of EDGE

math operation IS_MSF (msf: GRAPH g: GRAPH): boolean

definition (* true iff msf is an MSF of g *)

interface

type Spanning_Forest_Machine_State
is modeled by (edges: GRAPH insertion_phase: boolean)

exemplar m
initialization
  ensures m = (empty_set, true)
operation Change_To_Insertion_Phase (  
    alters      m: Spanning_Forest_Machine_State  
  ) 
  requires    not m.insertion_phase  
  ensures     m = (empty_set, true)

operation Insert (  
    alters      m: Spanning_Forest_Machine_State  
    consumes    v1: Integer  
    consumes    v2: Integer  
    consumes    w: Integer  
  ) 
  requires    m.insertion_phase and  
              1 <= v1 <= max_vertex and  
              1 <= v2 <= max_vertex and  
              w > 0  
  ensures     IS_MSF (m.edges,  
                      #m.edges union  
                      {(#v1, #v2, #w)}) and  
                      m.insertion_phase

operation Change_To_Extraction_Phase (  
    alters      m: Spanning_Forest_Machine_State  
  ) 
  requires    m.insertion_phase  
  ensures     m = (#m.edges, false)

operation Extract (  
    alters      m: Spanning_Forest_Machine_State  
    produces    v1: Integer  
    produces    v2: Integer  
    produces    w: Integer  
  ) 
  requires    m.edges /= empty_set and  
              not m.insertion_phase  
  ensures     (v1, v2, w) is in  
              #m.edges and  
              m = (#m.edges without  
                     {(#v1, #v2, #w)}, false)

operation Size (  
    preserves   m: Spanning_Forest_Machine_State  
  ): Integer  
  ensures     Size = |m.edges|
operation Is_In_Insertion_Phase (preserves m: Spanning_Forest_Machine_State):
  ensures Is_In_Insertion_Phase iff m.insertion_phase
end Spanning_Forest_Machine_Template
realization Kruskal_Amortized
   for Spanning_Forest_Machine_Template

collection

  global context

  ...

  parametric context

  ...

  local context

  type Edge is record
      vertex1: Integer
      vertex2: Integer
      weight: Integer
   end record

  facility Sorting_Machine_Facility is
      Sorting_Machine_Template
      (Edge, EDGES_ARE_ORDERED)
   realized by Heapsort_Embedding (...

  facility Coalesceable_Equivalence_Relation_Facility is
      Coalesceable_Equivalence_Relation_Template (max_vertex)
   realized by Disjoint_Set (...

  type Spanning_Forest_Machine_State_Reps is record
      graph_edges: Sorting_Machine_State
      are_connected: Coalesceable_Equivalence_Relation
      num_spanning_edges: Integer
   end record

  ...

interface

  type Spanning_Forest_Machine_State
   is represented by
      Spanning_Forest_Machine_State_Reps
  convention       (* rep invariant *)
  correspondence   (* representation-abstraction relation *)
operation Change_To_Insertion_Phase (  
alters m: Spanning_Forest_Machine_State  
)  
new_rep: Spanning_Forest_Machine_State_Rep  
begin  
m.rep := new_rep  
end Change_To_Insertion_Phase

operation Insert (  
alters m: Spanning_Forest_Machine_State  
consumes v1: Integer  
consumes v2: Integer  
consumes w: Integer  
)  
begin  
if not Are_Equivalent (m.rep.are_connected, v1, v2)  
then  
Make_Equivalent (m.rep.are_connected, v1, v2)  
m.rep.num_spanning_edges := m.rep.num_spanning_edges + 1  
end if  
Insert (m.rep.graph_edges,(v1,v2,w))  
end Insert

operation Change_To_Extraction_Phase (  
alters m: Spanning_Forest_Machine_State  
)  
new_equivalence_relation: Coalesceable_Equivalence_Relation  
begin  
Change_To_Extraction_Phase (m.rep.graph_edges)  
m.rep.are_connected := new_equivalence_relation  
end Change_To_Extraction_Phase
operation Extract (  
   alters m: Spanning_Forest_Machine_State  
   produces v1: Integer  
   produces v2: Integer  
   produces w: Integer  
)
begin  
   loop  
   maintaining (* loop invariant *)  
   Extract (m.rep.graph_edges, (v1, v2, w))  
   if not Are_Equivalent  
      (m.rep.are_connected, v1, v2)  
      then  
         Make_Equivalent  
         (m.rep.are_connected, v1, v2)  
         m.rep.num_spanning_edges :=  
         m.rep.num_spanning_edges - 1  
         exit  
      end if  
   end loop  
end Extract  
  
operation Size (  
   preserves m: Spanning_Forest_Machine_State  
): Integer  
begin  
   return m.rep.num_spanning_edges  
end Size  
  
operation Is_In_Insertion_Phase (  
   preserves m: Spanning_Forest_Machine_State  
): Boolean  
begin  
   return Is_In_Insertion_Phase  
   (m.rep.graph_edges)  
end Is_In_Insertion_Phase  
end Spanning_Forest_Machine_Template