This appendix lists some of the more important EXPRESS-2 functions that were developed for building information models of the composite ribs reported in Chapter 7. The functions have been categorized by the modeling activities that they enable. The three categories are; mouse hole building, decision graph solving and process model selection. These listing are intended to give the reader a feel for the nature of the EXPRESS-2 language and its usage in these models. Hence, a number of simple or insignificant functions have been omitted.

### F.1 Mouse Hole Build Functions

```plaintext
FUNCTION build_mouse_hole( hole : mouse_hole; inrib : rib) : BOOLEAN;
LOCAL
  depth : REAL;  width : REAL;  base_face : face;  base_bound : face_bound;
  base_poly : poly_loop;  c : REAL;  h : REAL;  xloc : REAL;  mouse_center : point;
  cut : poly_loop;  frame : poly_loop;  cps : BAG OF UNIQUE point;
  cp1 : point;  cp2 : point;  top_is_1 : BOOLEAN;
END_LOCAL;

conout('Starting algorithm');
-- Get the dimensionalised mouse hole and rib parameters
  c:= inrib.c;  h:= inrib.h_on_c * c;  xloc:= hole.x_on_c * c;
  depth:= hole.depth_on_c * c;  width:=hole.width_on_c * c;

-- Get the poly_loop of the perimeter of the input rib
  base_face:=inrib.geometry;
  base_bound:=base_face.bounds[1];
  base_poly:=base_bound.bound;
  IF (NOT(EXISTS(base_poly))) THEN conout('Error extracting base poly_loop from input rib.');  RETURN (FALSE); END_IF;

-- Get the mouse hole center point by crossing the pl1 and frame poly_loops and selecting top or bottom point
  frame:=create_frame(c,xloc,h);  -- a rectangle whos right side interestects the rib top and bottom
  cps:=cross_polys(base_poly,frame);  -- cps should contain exactly two points
  IF (NOT(SIZEOF(cps)=2)) THEN conout('Error setting mousehole center point, check rib profile at specified x_on_c.');
  RETURN (FALSE); END_IF;
  cp1:=cps[1];  cp2:=cps[2];
  top_is_1:= (cp1.y) > (cp2.y);

-- Logic to determine the desired cross point
  IF (hole.top XOR top_is_1) THEN mouse_center:=cp2;
```

APPENDIX F
EXPRESS-2 Functions

---

166
ELSE mouse_center:=cp1; END_IF;

    -- Create the cut and put a reference of it into the hole entity
    cut:=create_mouse_shape(mouse_center,depth,width);
    IF (NOT(EXISTS(cut))) THEN conout('Error making mouse hole shaped poly_loop.');
    RETURN (FALSE); END_IF;
    hole.cutter:=cut;conout('Mouse hole sucessfully built');
    RETURN (TRUE);
END_FUNCTION;

-- Creates a new poly_loop with a shape that can be used for both top and bottom
mouse_holes
FUNCTION create_mouse_shape(center:point; d:REAL; w:REAL):poly_loop;
LOCAL
  p : point;  tbag : BAG OF UNIQUE point;  shape : poly_loop;
END_LOCAL;
  p:=point( center.x - w/2 , center.y - d + w/4  , 0 );  dropin(p,tbag);
  p:=point( center.x - w/4 , center.y - d  , 0 );  dropin(p,tbag);
  p:=point( center.x + w/4 , center.y - d  , 0 );  dropin(p,tbag);
  p:=point( center.x + w/2 , center.y - d + w/4  , 0 );  dropin(p,tbag);
  p:=point( center.x + w/2 , center.y + d - w/4  , 0 );  dropin(p,tbag);
  p:=point( center.x + w/4 , center.y + d  , 0 );  dropin(p,tbag);
  p:=point( center.x - w/4 , center.y + d  , 0 );  dropin(p,tbag);
  p:=point( center.x - w/2 , center.y + d - w/4  , 0 );  dropin(p,tbag);
  IF (NOT(SIZEOF(tbag)=8)) THEN conout('Error creating points that form the mouse_hole
shape.');    RETURN (?);  END_IF;
  shape:=poly_loop(tbag);
  RETURN (shape);
END_FUNCTION;

-- Takes a bite out of face 'f'. Returns '?' in the following conditions:
--      1. 'f' and 'pl' are not co-planar
--      2. 'pl' and exterior bounding polygon of 'f' don't cross
FUNCTION bite( f:face; cutter:poly_loop ):face;
LOCAL
  pl1 : poly_loop;     pl2 : poly_loop;     face_outer : face_bound;     crosspts : BAG OF
  UNIQUE point;
  new_perim : poly_loop;     new_bound : face_bound;     new_face : face;     cp:point;
END_LOCAL;
  conout('Starting face bite algorithm');
  face_outer:=f.bounds[1];
  pl1:=poly_loop(face_outer.bounds.polygon); -- Work with the exterior polygon of the face
  pl2:=poly_loop(cutter.polygon); -- work with a copy of 'cutter' in case of
errors
  IF NOT exists(pl1) THEN RETURN (?); END_IF;
  crosspts:=cross_polys(pl1.pl2); -- Get the cross-over points
  IF (SIZEOF(crosspts)=0) THEN
    conout('Error, cutting poly_loop does not cross the perimeter of the base face.'); RETURN
    (?) END_IF;
  REPEAT FOREACH cp IN crosspts;
  seg_poly(pl1,cp); -- Segment the pols at the cross-over points
  seg_poly(pl2,cp);
167
F.2 Decision Graph Solver Functions

FUNCTION propagate_switches(pm : process_model) : BOOLEAN;
LOCAL
    s : switch;  changed : BOOLEAN;
    remaining : BAG OF UNIQUE switch;
    lines : BAG OF STRING;
    mkb : manufacturing_knowledge_base;
END_LOCAL;

    mkb:=pm.knowledge;
    IF (NOT(EXISTS(mkb))) THEN
        conout('Error, Process model has no manufacturing knowledge base defined.);
        RETURN (?);
    END_IF;
    IF (NOT(EXISTS(mkb.completion_switch))) THEN
        conout('Error, no completion switch is defined in knowledge base.);
        RETURN (?);
    END_IF;
    IF (SIZEOF(pm.start_switches*mkb.completion_switch)=1) THEN
        conout('Error, completion switch is included in the start switches for process model.);
        RETURN (?);
    END_IF;
    IF (sizeof(pm.start_switches)=0) THEN
        conout('Error, no start switches in bag.);
        RETURN (?);
    END_IF;
    pm.report:=pm.report+['1. Determination of manufacturing switches',''];
    remaining:=mkb.all_switches;
    REPEAT FOREACH s IN remaining; s.active:=unknown; END_REPEAT;
    changed:=true;
    REPEAT WHILE (changed);
        conout('Starting new loop through all switches');
        changed:=false;
        REPEAT FOREACH s IN mkb.all_switches;
            IF (s.active=unknown) THEN
                IF (determine_state(s,pm)) THEN
                    IF (NOT(pullout(s,remaining))) THEN
                        conout('Error pulling out switch "'+s.name+'" from remaining bag.);
                        RETURN (?);
                    END_IF;
                END_IF;
                changed:=true;
            END_IF;
        END_REPEAT;
    END_REPEAT;
RETURN (True);
END_IF;
IF ((s.active=false) AND (sizeof(pm.start_switches*s)=1) ) THEN
   conout('Error, switch "' + s.name + '" is a start switch that could not be activated.');
   report_switch(s);
   RETURN (false);
END_IF;
END_IF;
END_REPEAT;
IF (SIZEOF(remaining)=0) THEN
   IF (mkb.completion_switch.active=false) THEN
      conout('Error, completion switch is inactive.');
      report_switch(mkb.completion_switch);
      RETURN (false);
   ELSE
      conout('All switches have been sucessfully determined.');
      pm.report:=pm.report+["","All switches have been sucessfully determined."];
      RETURN (true);
   END_IF;
END_IF;
END_REPEAT;
conout('Unable to complete switch propagation.);
conout('Error, '+to_string(sizeof(remaining))+" switch/es in an unknown state:');
REPEAT FOREACH s IN remaining;
   report_switch(s);
   END_REPEAT;
RETURN (false);

END_FUNCTION;

FUNCTION determine_state(test : switch; pm : process_model) : BOOLEAN;
LOCAL
   s : switch;
   or_satisfied : BOOLEAN;
END_LOCAL;
IF (NOT(test.active=unknown)) THEN RETURN (true); END_IF;
IF ( (SIZEOF(test.and_switches)=0) AND
     (SIZEOF(test.or_switches)=0) AND
     (SIZEOF(test.not_switches)=0) AND
     (SIZEOF(pm.start_switches*test)=0) ) THEN
   test.active:=false;
   pm.report:=pm.report+[" ",""+'+test.name+'" is INACTIVE since it has no dependant switches
and is not a start switch.' ];
   RETURN (true); -- exit with a no-dependencies failure
END_IF;
REPEAT FOREACH s IN test.and_switches;
   IF (s.active=unknown) THEN RETURN (false); END_IF;
   IF (NOT(s.active) AND (SIZEOF(pm.start_switches*s)=0)) THEN
      test.active:=false;
      pm.report:=pm.report+[" ",""+'+test.name+'" is INACTIVE since AND switch "'+s.name+'" is
inactive and not a start switch.' ];
      RETURN (true); -- exit with an AND failure
   END_IF;
END_REPEAT;
-- getting to here means all of the and's were true.
IF (SIZEOF(test.or_switches)>0) THEN
   or_satisfied:=false;
   REPEAT FOREACH s IN test.or_switches;
IF (s.active=unknown) THEN RETURN (false); END_IF;
 IF (s.active=true OR (SIZEOF(pm.start_switches*s)=1) ) THEN or_satisfied:=true;
 END_IF;
END_REPEAT;
ELSE
 or_satisfied:=true;
ENDIF;
IF (NOT(or_satisfied)) THEN
 test.active:=false;
pm.report:=pm.report+('’”+test.name+”’ is INACTIVE since none of its OR switches were
active or start switches.’);
 RETURN (true); -- exit with an OR inactive switch
ENDIF;
-- getting to here means both the ANDs and the ORs were satisfied
REPEAT FOREACH s IN test.not_switches;
 IF (s.active=unknown) THEN RETURN (false); END_IF;
 IF (s.active=true OR (SIZEOF(pm.start_switches*s)=1) ) THEN
 test.active:=false;
 pm.report:=pm.report+('’”+test.name+”’ is INACTIVE since NOT switch ”’”+s.name+”’ was
active or a start switch.’);
 RETURN (true); -- exit with a NOT inactive switch
ENDIF;
END_REPEAT;
test.active:=true;
pm.report:=pm.report+('’”+test.name+”’ is ACTIVE.’);
RETURN (true); -- exit with the test being sucessfully switched on!
END_FUNCTION;

F.3 Process Model Selector Functions

FUNCTION build_process_model(mod : process_model) : BOOLEAN;
LOCAL
 stages : BAG OF UNIQUE stage;
s : stage;
sw : switch;
p : process;
str : STRING;
total_time : REAL;
END_LOCAL;

conout(’Starting algorithm to build the process model.’);
IF (NOT(EXISTS(mod.knowledge))) THEN conout(’Error, model does not reference a
manufacturing knowledge base.’); RETURN (?); END_IF;
 IF (NOT(EXISTS(mod.part_properties))) THEN conout(’Error, model does not reference a
panel summary.’); RETURN (?); END_IF;
IF (SIZEOF(mod.start_switches)<1) THEN conout(’Error, model does not have any start
switches defined.’); RETURN (?); END_IF;

-- Indicate the part geometric properties and
mod.report:=['Process Model Report: ’+mod.name];
mod.report:=mod.report+’ ’+to_string(mod.part_properties.area));
mod.report:=mod.report+’ ’+to_string(mod.part_properties.perimeter))};
-- Determine manufacturing switches
cout('Determining active switches ...');
IF (NOT(propagate_switches(mod))) THEN cout('Error propagating switches, check starting switches, completion switch and switch logic.'); RETURN (FALSE); END_IF;

-- Build the process stages
cout('Building the manufacturing stages ...');
stages:=build_stages(mod);
IF (NOT(EXISTS(stages)) OR (SIZEOF(stages)<1) ) THEN cout('Error processing manufacturing switches, no manufacturing stages were defined.'); RETURN (FALSE); END_IF;
conout('Process stages built sucessfully, generating report.);
mod.report:=mod.report+'';
mod.report:=mod.report+''In this model, the following '+ to_string(sizeof(stages)) + ' stages were used:'');
total_time:=0;
REPEAT FOREACH s IN stages;
total_time:=total_time+s.time;
str:=s.type.name;
IF (EXISTS(s.type.details)) THEN  str:=str+'' - '' + s.type.details + ''; END_IF; -- details may not exist yet
mod.report:=mod.report+str;
END_REPEAT;

-- Show the process time breakdown
cout('Generating process time breakdown for the report.);
mod.report:=mod.report+'';
mod.report:=mod.report+'';
mod.report:=mod.report+''Process times were estimated with the PCAD database.'';
mod.report:=mod.report+''The total time was '+to_string(total_time/60/60)+' hours.'';
mod.report:=mod.report+''A detailed process time breakdown is given below.'';
REPEAT FOREACH s IN stages;
   mod.report:=mod.report+'' '+s.type.name+'' : '+to_string(s.time/60/60)+' hours'';
   REPEAT FOREACH p IN s.processes;
      mod.report:=mod.report+'' '+p.type.description + ' : '+ to_string(p.time/60/60)+' hours'';
   END_REPEAT;
END_REPEAT;
END_FUNCTION;

FUNCTION all_known( b : BAG OF UNIQUE switch) : BOOLEAN;
LOCAL
   s : switch;
END_LOCAL;
REPEAT FOREACH s IN b;
   IF (s.active=unknown) THEN RETURN (false); END_IF;
END_REPEAT;
RETURN (true);
END_FUNCTION;

FUNCTION build_stages( mod : process_model) : BAG OF UNIQUE stage;
LOCAL
  stages : BAG OF UNIQUE stage;
  new_stage : stage;
  type : stage_type;
END_LOCAL;

REPEAT FOREACH type IN mod.knowledge.stage_types;
  IF (activate_stage(type,mod)) THEN
    conout('Stage type "'+type.name+'" was activated, building stage into process model...');
    new_stage:=build_stage(type,mod.part_properties);
    IF (NOT(EXISTS(new_stage))) THEN conout('Error building stage '+type.name+'.');
    RETURN (?); END_IF;
  endin;
END_REPEAT;
conout('Successfully built '+to_string(sizeof(stages))+' stages.');
RETURN (stages);
END_FUNCTION;

FUNCTION activate_stage(type : stage_type; pm : process_model):BOOLEAN;
LOCAL
  s : switch;
  or_satisfied : BOOLEAN;
END_LOCAL;

-- AND switches
REPEAT FOREACH s IN type.and_switches;
  IF (NOT(s.active=true)) THEN RETURN (false); END_IF;
END_REPEAT;

-- OR switches
IF (SIZEOF(type.or_switches)=0) THEN or_satisfied:=true;
ELSE
  or_satisfied:=false;
  REPEAT FOREACH s IN type.or_switches;
    IF (s.active=true) THEN or_satisfied:=true; END_IF;
  END_REPEAT;
END_IF;

-- NOT switches
REPEAT FOREACH s IN type.not_switches;
  IF (s.active=true) THEN RETURN (false); END_IF;
END_REPEAT;
IF (or_satisfied) THEN RETURN (true); ELSE RETURN (false); END_IF;
END_FUNCTION;

FUNCTION build_stage( type : stage_type; props : panel_summary):stage;
LOCAL
  stage_time : REAL;
  new_proc : process;
  procs : BAG OF process;
  new_stage : stage;
  pcad : PCAD_process;
  new_time : REAL;
END_LOCAL;

  stage_time:=0;
REPEAT FOREACH pcad IN type.steps;
   new_proc:= process(pcad.description,pcad,?);
   new_time:=eval_process_time(new_proc,props);
   IF (NOT(EXISTS(new_time))) THEN conout('Error evaluating process time for process of type ' +type.name+'); RETURN (?); END_IF;
   stage_time:=stage_time+new_time;
   dropin(new_proc,procs);
END_REPEAT;
new_stage:=stage(type.name,type,procs,stage_time);
RETURN (new_stage);

END_FUNCTION;