```
(************************************************************************
************************************************************************
********
******** Concrete Categories, base definitions
******** Language SML/NY
******** Author M. Bertol (c)
********
******** Release: Tue Aug 13 11:35:40 MET DST 1996                                10
********
************************************************************************
************************************************************************)



(*** datatype for a category ***)

datatype('o, 'a)Cat = cat of ('a->'o) * ('a->'o) * ('o->'a) * ('a*'a->'a);      cat
                                                                                20
fun dom( cat(d, _,_,_)) = d;                                                    dom
fun cod( cat(_, c, _,_)) = c;                                                   cod
fun id( cat(_,_, i,_)) = i;                                                     id
fun compose( cat(_,_,_, c)) = c;                                                compose


(*** universal construction: initial object ***)

type ('o,'a)InitialObj = 'o * ('o -> 'a);                                       type
                                                                                30
fun init_obj( (ob, _): ('o,'a)InitialObj) = ob;                                 init_obj
fun init_univ( (_, ar): ('o,'a)InitialObj) = ar;                                init_univ
fun iso_initial((a,univ_a), (a',univ_a')) = (univ_a(a'),univ_a'(a));


(*** datatype for functors ***)

datatype ('oA, 'aA, 'oB, 'aB)Functor =
                    cat_functor of ('oA, 'aA)Cat *                              cat_functor
                                   ('oA -> 'oB) * ('aA -> 'aB) *                40
                                   ('oB, 'aB)Cat;

fun domain(cat_functor(D,_,_,_)) = D;                                           domain
fun range(cat_functor(_,_,_,R)) = R;                                            range
fun apply_obj(cat_functor(_,fobj,_,_)) = fobj;                                  apply_obj
fun apply_arr(cat_functor(_,_,farr,_)) = farr;                                  apply_arr


(*** duality operations ***)
                                                                                50
fun dual_Cat(cat(s,t,i,c)) = cat(t, s, i, fn (g,f) => c(f,g));                  dual_Cat

fun dual_Fun(cat_functor(A,fo,fa,B)) =                                          dual_Fun
                    cat_functor(dual_Cat(A), fo, fa, dual_Cat(B));



(*** universal construction: coproduct ***)

type ('o, 'a)Coproduct_CoCone = ('o * 'a * 'a) * ('o * 'a *'a -> 'a);           type
                                                                                61
type ('o, 'a)CoProduct = 'o * 'o -> ('o, 'a)Coproduct_CoCone;                   type
```

```
(*** universal construction: coequalizer ***)

type ('o, 'a)CoEqualizer = 'a * 'a -> ('o * 'a) * ('o * 'a -> 'a);                    type
                                                                                      70

(*** universal construction: pushout ***)

type ('o, 'a)Pushout = 'a * 'a -> ('a * 'a) * ('a * 'a -> 'a);                        type


(*** datatype for diagram graphs ***)

exception undefined_Edge;                                                             80
exception undefined_Vertex;

datatype ('v,'e)multiGraph = graph of ('v list * 'e list *                            graph
                                ('e -> 'v) * ('e -> 'v));

val nil_graph = graph([], [],                                                         nil'graph
                  fn _ => raise undefined_Edge,
                  fn _ => raise undefined_Edge);

fun is_nil_graph(graph([],_,_,_)) = true                                              is'nil'graph
  | is_nil_graph(graph(_,_,_,_)) = false;                                             91


(*** datatype for diagrams ***)

exception undefined;

datatype ('n, 'e, 'o, 'a)Diagram =                                                    diagram
                diagram of ('n, 'e)multiGraph * ('n -> 'o) * ('e -> 'a);              100

fun is_nil_diagram(diagram(g,_,_)) = is_nil_graph(g);                                 is'nil'diagram
val nil_diagram = diagram(nil_graph,                                                  nil'diagram
                     fn x => raise undefined,
                     fn y => raise undefined);
fun obj_at_node(diagram(_, objs, _)) = objs;                                          obj'at'node
fun arr_at_edge(diagram(_, _, arrs)) = arrs;                                          arr'at'edge

fun nodes(diagram(graph(vertices,_,_,_),_,_)) = vertices;                             nodes
                                                                                      110
fun remove_vertex(graph(vertices,edges,source,drain),v) =                             remove'vertex
   let fun remove(x,[]) = []
        | remove (x,y::l) = if x = y then l
                            else y::remove(x,l);
       fun remove_incident(v,[]) = []
         | remove_incident(v,e::l) = if (drain(e)=v orelse source(e)=v)
                                     then remove_incident(v,l)
                                else remove_incident(v,l)
   in graph(remove(v,vertices),remove_incident(v,edges),source,drain)
   end;                                                                               120

fun reduce_diagram(diagram(g, objs, arrs), n) =                                       reduce'diagram
   diagram(remove_vertex(g,n), objs, arrs);
```

```
(*** datatype for cocones ***)

datatype ('n, 'e, 'o, 'a)CoCone =                                          cocone
   cocone of 'o * ('n, 'e, 'o, 'a)Diagram * ('n -> 'a);
datatype ('n, 'e, 'o, 'a)CoCone_Arrow =                                    cocone_arrow
   cocone_arrow of ('n, 'e, 'o, 'a)CoCone * 'a * ('n, 'e, 'o, 'a)CoCone;   131

fun co_apex(cocone( ob, _, _)) = ob;                                       co'apex
fun sides(cocone(_,_, fu)) = fu;                                           sides
fun base(cocone(_, di, _)) = di;                                           base

fun co_apex_arrow(cocone_arrow(_, ar, _)) = ar;                            co'apex'arrow


(*** datatype for limits ***)                                             140

type ('n, 'e, 'o, 'a)Colimiting_CoCone =                                   type
   ('n, 'e, 'o, 'a)CoCone *
   (('n, 'e, 'o, 'a)CoCone -> ('n, 'e, 'o, 'a)CoCone_Arrow);

type ('n, 'e, 'o, 'a)CoLimit =                                             type
   ('n, 'e, 'o, 'a)Diagram -> ('n, 'e, 'o, 'a)Colimiting_CoCone;


(*** datatype for a complete category ***)                                150

datatype ('n, 'e, 'o, 'a)CoComplete_Cat =                                  complete'cat
   cocomplete_cat of ('o, 'a)Cat * ('n, 'e, 'o, 'a)CoLimit;


(*** datatype for a cocontinous functor ***)

datatype ('n, 'e, 'oA, 'aA, 'oB, 'aB)CoContinous_Functor =                 cocontinous'functor
   cocontinous_functor of ('oA, 'aA, 'oB, 'aB)Functor *
   (('n, 'e, 'oA, 'aA)Colimiting_CoCone -> ('n, 'e, 'oB, 'aB)Colimiting_CoCone);   160


(*** functions for determining colimits ***)

fun finite_coproduct(C,initial,b_coprod) =                                 finite'coproduct
   fn d => if is_nil_diagram(d)
           then let val (i, i_univ) = initial;
                    val i_cocone = cocone(i,
                                          nil_diagram,
                                          fn x => raise undefined)         170
                in (i_cocone,
                    fn c1 => cocone_arrow(i_cocone, i_univ(co_apex(c1)), c1))
                end
         else let val n::_ = nodes(d) ;
                  val (c,univc) = finite_coproduct(C,initial,b_coprod)
                                       (reduce_diagram(d,n));
                  val ((b,f,g), univcp) = b_coprod(co_apex(c),obj_at_node(d)(n));
                  val result_cocone = cocone(b,d,
                                        fn m => if m=n then g
                                                else compose(C)(f,sides(c)(m)));   180
                  val universal = fn c1 => let val u = co_apex_arrow(univc(c1))
                                             val v = univcp(co_apex(c1),
                                                            u,
                                                            sides(c1)(n))
                                           in cocone_arrow(result_cocone, v, c1)
                                           end
```

```
            in (result_cocone, universal)
            end;

                                                                          190
fun add_edge (C,coeq) =                                                   add_edge
   fn ((c,univ), e) => let val diagram(g, fo, fa) = base(c);
                       val graph(vertices,edges,s,t) = g;
                       val ((b,h),ce_univ) = coeq(sides(c)(s(e)),
                                                  compose(C)(sides(c)(t(e)),
                                                             fa(e)));
                       val result_graph = graph(vertices,e::edges,s,t);
                       val result_diagram = diagram(result_graph, fo, fa);
                       val result_cocone = cocone(b,
                                                  result_diagram,          200
                                                  fn n => compose(C)
                                                              (h,sides(c)(n)));
                       val universal =
                           fn c1 => let val u = co_apex_arrow(univ(c1))
                                        val v = ce_univ(co_apex(c1),u)
                                    in cocone_arrow(result_cocone, v, c1)
                                    end
                  in (result_cocone, universal)
                  end;
                                                                          210

(*** categorie with initials, coproducts, and coequalizers ***)

datatype ('o, 'a)IO_CP_CE_Cat = io_cp_ce_cat of ('o, 'a)Cat *             io_cp_ce_cat
                                        ('o, 'a)InitialObj *
                                        ('o, 'a)CoProduct *
                                        ('o, 'a)CoEqualizer;


(*** function for computing the limit objects ***)                        220

fun finite_colimit(cC as io_cp_ce_cat(C, init, b_coprod, coeq)) =         finite_colimit
   fn (d as diagram(graph(nodes,edges,s,t), fo, fa)) =>
   if edges=[] then finite_coproduct(C, init, b_coprod)(d)
   else let val e::edges' = edges
            val d1 = diagram(graph(nodes,edges',s,t), fo, fa)
        in add_edge(C,coeq)((finite_colimit cC d1), e)
        end;

                                                                          230
(*** datatypes for the dual constructions ***)


(*** datatype for the product ***)

type ('o, 'a)Product = 'o * 'o -> ('o * 'a * 'a) * ('o * 'a * 'a -> 'a);  type



(*** datatype for a cone ***)                                             240

datatype ('n, 'e, 'o, 'a)Cone =                                           cone
   cone of 'o * ('n, 'e, 'o, 'a)Diagram * ('n -> 'a);

datatype ('n, 'e, 'o, 'a)Cone_Arrow =                                     cone_arrow
   cone_arrow of ('n, 'e, 'o, 'a)Cone * 'a * ('n, 'e, 'o, 'a)Cone;
```

```
(*** datatype for a limit ***)                                                    250

type ('n, 'e, 'o, 'a)limiting_Cone =                                             type
    ('n, 'e, 'o, 'a)Cone * (('n, 'e, 'o, 'a)Cone -> ('n, 'e, 'o, 'a)Cone_Arrow);

type ('n, 'e, 'o, 'a)Limit = ('n, 'e, 'o, 'a)Diagram -> ('n, 'e, 'o, 'a)limiting_Cone;  type


(*** datatype for a complete category, realized via dualization ***)
                                                                                 260
datatype ('n, 'e, 'o, 'a)Complete_Cat =                                          complete_cat
    complete_cat of ('o, 'a)Cat * ('n, 'e, 'o, 'a)Limit;

fun dual_graph(graph(N,E,s,t)) = graph(N,E,t,s);
fun dual_diagram(diagram(g,nm,em)) = diagram(dual_graph(g),nm,em);

fun dual_cone(cocone(a,D,f),univ) =
    let val result_cone = cone(a,dual_diagram(D),f);
        val universal = fn (c1 as cone(a1,D1,f1)) =>
                        let val c2 = cocone(a1,dual_diagram(D1),f1)               270
                        in cone_arrow(c1,
                                      co_apex_arrow(univ c2),
                                      result_cone)
                        end
    in (result_cone,universal)
    end


fun dual_colimit(colim) = fn D => dual_cone(colim(dual_diagram(D)));              dual_colimit
                                                                                 280
(*** end of definition ***)
```