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


use"cat.sml";


(*** datatypes for comma categories ***)
                                                                                20
datatype ('oA,'aA,'oC,'aC,'oB,'aB)Comma_Arrow =                                 datatype
                 comma_arrow of ('oA * 'aC * 'oB) *                             comma_arrow
                                ('aA * 'aB) *
                                ('oA * 'aC * 'oB);

fun comma_cat(L,R) = let val A = domain(L);                                     comma_cat
                         val B = domain(R)
                     in cat(fn comma_arrow(Y,_,_) => Y,
                            fn comma_arrow(_,_,Y) => Y,
                            fn (a,b,c) => comma_arrow((a,b,c),                  30
                                                      (id(A)(a),id(B)(c)),
                                                      (a,b,c)),
                            fn (comma_arrow(_,(f,g),Z),
                                comma_arrow(Y,(h,j),_)) =>
                                       comma_arrow(Y,
                                                   (compose(A)(f,h),compose(B)(g,j)),
                                                   Z))
                     end;

                                                                                40
(*** retract operations for comma categories ***)

fun left_obj(l,_,_) = l;                                                        left_obj

fun left_arr(comma_arrow((Aobj, Carr, Bobj),                                    left_arr
                         (Aarr, Barr),
                         (Aobj', Carr', Bobj'))) = Aarr;
fun left_Diag(diagram(g,obj,arr)) = diagram(g,                                  left'Diag
                                    fn ver => left_obj(obj(ver)),
                                    fn edg => left_arr(arr(edg)) );             50
fun left_CoCone(cocone(c,d,marr)) = cocone(left_obj(c),                         left'CoCone
                                   left_Diag(d),
                                   fn ver => left_arr(marr(ver)));

fun right_obj(_,_,r) = r;                                                       right'obj
fun right_arr(comma_arrow((Aobj, Carr, Bobj),                                   right'arr
                          (Aarr, Barr),
                          (Aobj', Carr', Bobj'))) = Barr;
fun right_Diag(diagram(g,obj,arr)) = diagram(g,                                 right_Diag
                                     fn ver => right_obj(obj(ver)),             60
                                     fn edg => right_arr(arr(edg)) );
fun right_CoCone(cocone(c,d,marr)) = cocone(right_obj(c),                       right_CoCone
```

```
                                right_Diag(d),                                  right_Diag
                                fn ver => right_arr(marr(ver)));


(*** necessary extensions of the functor datatype ***)


fun apply_Fun_Diag(Fct,d) = let val diagram(g,obj,arr) = d;                      apply_Fun_Diag
                            val obj' = fn ver => (apply_obj(Fct))(obj(ver));    71
                            val arr' = fn edg => (apply_arr(Fct))(arr(edg))
                        in diagram(g, obj',arr')
                        end;


fun apply_Fun_CoCone(Fct, cocone(c,d,marr)) = cocone(apply_obj(Fct)(c),          apply_Fun_CoCone
                                              apply_Fun_Diag(Fct,d),
                                              fn ver =>
                                                    apply_arr(Fct)(marr(ver)));  80

fun apply_Fun_CoConeArrow(Fct, cocone_arrow(Acone,arr,Bcone)) =                  apply_Fun_CoConeArrow
              cocone_arrow(apply_Fun_CoCone(Fct,Acone),
                           apply_arr(Fct)(arr),
                           apply_Fun_CoCone(Fct,Bcone));


(*** functor have to preserve colimits ***)

fun preserve(cocontinous_functor(Fct,translation)) = translation;                preserve
fun real(cocontinous_functor(Fct,translation)) = Fct;                            real
                                                                                 92

(*** lifting of the colimits into the comma category ***)


fun lift_colimit(colimA,contL,R,colimB) =                                        lift_colimit
    fn D => let val (c_A,univ_A) = colimA(left_Diag(D));
            val (c_B,univ_B) = colimB(right_Diag(D));
            val (Lc,Luniv) = preserve(contL)(c_A,univ_A);                        100
            val Rc = apply_Fun_CoCone(R,c_B);
            val c1 = cocone(co_apex(Rc),
                            base(Lc),
                            fn n => let val (_,f_n,_) = obj_at_node(D)(n)
                                    in compose(range(real(contL)))
                                       (sides(Rc)(n),f_n)
                                    end);
            val f = co_apex_arrow(Luniv c1);
            val colim_obj = (co_apex(c_A),f,co_apex(c_B))
            val result_cocone = cocone(colim_obj,                                110
                                       D,
                                       fn n =>
                                          comma_arrow(obj_at_node(D)(n),
                                                      (sides(c_A)(n),sides(c_B)(n)),
                                                      colim_obj));
            val universal = fn c =>
                              let val uA = co_apex_arrow(univ_A(left_CoCone(c)))
                                  val uB = co_apex_arrow(univ_B(right_CoCone(c)))
                              in cocone_arrow(result_cocone,
                                              comma_arrow(colim_obj,             120
                                                          (uA,uB),
                                                          co_apex(c)),
                                              c)
                              end
```

```
        in (result_cocone, universal)
        end


(*** cocomplete comma categories ***)
                                                                          130
fun cocomplete_comma_cat(cA,cB) (contL,R) =                               cocomplete_comma_cat
        let val cocomplete_cat(_,colimA) = cA
            val cocomplete_cat(_,colimB) = cB
        in cocomplete_cat(comma_cat(real(contL),R),
                          lift_colimit(colimA,contL,R,colimB))
        end



(*** datatypes for functor categories ***)                                 140


datatype ('oA, 'aA, 'oB, 'aB)Nat_transform = nat_transform of ('oA, 'aA, 'oB, 'aB)Functor *
                                                ('oA -> 'aB) *            nat_transform
                                                ('oA, 'aA, 'oB, 'aB)Functor

fun id(A,cat(_,_,i,_)) F = nat_transform(F, fn a => i(apply_obj(F)(a)),F);

fun dotcomp(A,cat(_,_,_,comp)) =                                          dotcomp
        fn (nat_transform(_,beta,H),                                      150
            nat_transform(F,alpha,_)) =>
                nat_transform(F,fn a => comp(beta(a),alpha(a)),H);

fun Fun_comp(cat_functor(_,Fobj,Farr,cB),cat_functor(cA,Gobj,Garr,_)) =   Fun_comp
        cat_functor(cA,
                fn obj => Fobj(Gobj(obj)),
                fn arr => Farr(Garr(arr)),
                cB);

fun ringcomp(cat(_,_,_,comp)) =                                           ringcomp
                fn (nat_transform(G,beta,G'),                             161
                    nat_transform(F,alpha,F')) =>
                        nat_transform(Fun_comp(G,F),
                                fn a => comp(beta(apply_obj(F')(a)),
                                          apply_obj(G)(alpha(a))),
                                Fun_comp(G',F'));


(*** definition of functor categories ***)
                                                                          170
fun cat_of_functors(A,B) = cat(fn nat_transform(s,_,_) => s,              cat'of'functors
                        fn nat_transform(_,_,t) => t,
                        id(A,B),
                        dotcomp(A,B));
```