cross products of functors

Johan G. F. Belinfante
2012 January 11

```
In[1]:= SetDirectory["l:"]; << goedel.12jan09a
```

```
:Package Title: goedel.12jan09a                2012 January 9 at 9:55 a.m.

Loading takes about thirteen minutes, half that time due to builtin pauses.

It is now:  2012 Jan 11 at 12:24

Loading Simplification Rules

TOOLS.M is now incorporated in the GOEDEL program as of 2010 September 3

weightlimit = 40

Loading completed.

It is now:  2012 Jan 11 at 12:37
```

summary

In this notebook it is shown that if **t** is a functor from a category **u** to **v** and if **w** is a functor from a category **x** to **y**, then their cross product **t ⊗ w** is a functor from the direct product of **u** and **x** to the direct product of **v** and **y**. The assumptions that **u** and **x** be categories are not really needed, but only that they be subclasses of **(V × V) × V**.

Reference. The concept of product functor is introduced on page 36 in Mac Lane's book.

```
In[2]:= "Saunders Mac Lane, Categories for the
          Working Mathematician, Springer-Verlag, New York, 1971.";
```

derivation

The derivation is done by **AssertTest**, together with a slew of lemmas to simplify the results. The definition of **functor** has been wrapped with **case** to prevent it from being automatically expanded out:

```
In[3]:= case[functor[t, x, y]]
```

```
Out[3]= case[and[equal[domain[t], range[x]],
          FUNCTION[t], subclass[composite[t, x], composite[y, cross[t, t]]],
          subclass[image[t, ids[x]], ids[y]]]]
```

One can however force the definition to be expanded out by using **assert**. In addition, the **funpart** wrapper is used initially to eliminate the **FUNCTION** hypothesis, and **u** and **x** are replaced with their restrictions to **V × V** to eliminate the hypotheses that they be ternary relations.

Lemma.

```
In[4]:= or[equal[cart[domain[funpart[t]], domain[funpart[w]]],
           cart[image[u, cart[V, V]], image[x, cart[V, V]]]],
         not[functor[funpart[t], composite[u, id[cart[V, V]]], v]],
         not[functor[funpart[w], composite[x, id[cart[V, V]]], y]]] // AssertTest

Out[4]= or[equal[cart[domain[funpart[t]], domain[funpart[w]]],
           cart[image[u, cart[V, V]], image[x, cart[V, V]]]],
         not[functor[funpart[t], composite[u, id[cart[V, V]]], v]],
         not[functor[funpart[w], composite[x, id[cart[V, V]]], y]]] == True

In[5]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[6]:= or[subclass[image[funpart[t], ids[u]], ids[v]],
         not[functor[funpart[t], composite[u, id[cart[V, V]]], v]]] // AssertTest

Out[6]= or[not[functor[funpart[t], composite[u, id[cart[V, V]]], v]],
         subclass[image[funpart[t], ids[u]], ids[v]]] == True

In[7]:= (% /. {t → t_, u → u_, v → v_}) /. Equal → SetDelayed
```

Lemma.

```
In[8]:= Map[implies[and[subclass[composite[Id, u], x], subclass[composite[Id, v], y]], #] &,
          SubstTest[implies, subclass[r, s], subclass[composite[r, z], composite[s, z]],
           {r → cross[u, v], s → cross[x, y]}]] // MapNotNot // Reverse

Out[8]= or[not[subclass[composite[Id, u], x]], not[subclass[composite[Id, v], y]],
         subclass[composite[cross[u, v], z], composite[cross[x, y], z]]] == True

In[9]:= (% /. {u → u_, v → v_, x → x_, y → y_, z → z_}) /. Equal → SetDelayed
```

Theorem. A monotonicity result.

```
In[10]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
           not[implies[p1, p4]], {p1 → and[subclass[u, x], subclass[v, y]],
            p2 -> subclass[composite[Id, u], x], p3 -> subclass[composite[Id, v], y],
            p4 -> subclass[composite[cross[u, v], z], composite[cross[x, y], z]]}]] // Reverse

Out[10]= or[not[subclass[u, x]], not[subclass[v, y]],
          subclass[composite[cross[u, v], z], composite[cross[x, y], z]]] == True

In[11]:= or[not[subclass[u_, x_]], not[subclass[v_, y_]],
          subclass[composite[cross[u_, v_], z_], composite[cross[x_, y_], z_]]] := True
```

Lemma.

```
In[14]:= equal[composite[cross[v, y], TWIST,
           cross[cross[funpart[t], funpart[w]], cross[funpart[t], funpart[w]]]],
          composite[cross[composite[v, cross[funpart[t], funpart[t]]],
           composite[y, cross[funpart[w], funpart[w]]]], TWIST]] // AssertTest

Out[14]= equal[composite[cross[composite[v, cross[funpart[t], funpart[t]]],
           composite[y, cross[funpart[w], funpart[w]]]], TWIST], composite[cross[v, y], TWIST,
          cross[cross[funpart[t], funpart[w]], cross[funpart[t], funpart[w]]]] == True

In[19]:= composite[cross[composite[v_, cross[funpart[t_], funpart[t_]]],
           composite[y_, cross[funpart[w_], funpart[w_]]]], TWIST] := composite[cross[v, y],
          TWIST, cross[cross[funpart[t], funpart[w]], cross[funpart[t], funpart[w]]]]
```

Lemma.

```
In[20]:= SubstTest[implies, and[subclass[p, q], subclass[r, s]],
          subclass[direct[p, r], direct[q, s]], {p -> composite[funpart[t], u, id[cart[V, V]]],
           q -> composite[v, cross[funpart[t], funpart[t]]],
           r -> composite[funpart[w], x, id[cart[V, V]]],
           s -> composite[y, cross[funpart[w], funpart[w]]]}] // Reverse

Out[20]= or[not[subclass[composite[funpart[t], u, id[cart[V, V]]],
           composite[v, cross[funpart[t], funpart[t]]]]],
          not[subclass[composite[funpart[w], x, id[cart[V, V]]],
           composite[y, cross[funpart[w], funpart[w]]]], subclass[
           composite[cross[composite[funpart[t], u], composite[funpart[w], x]], TWIST],
           composite[cross[v, y], TWIST,
           cross[cross[funpart[t], funpart[w]], cross[funpart[t], funpart[w]]]]]] == True

In[21]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Lemma.

```
In[23]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
           not[implies[p1, p4]], {p1 → and[functor[funpart[t], composite[u, id[cart[V, V]]], v],
              functor[funpart[w], composite[x, id[cart[V, V]]], y]],
            p2 -> subclass[composite[funpart[t], u, id[cart[V, V]]],
              composite[v, cross[funpart[t], funpart[t]]]],
            p3 -> subclass[composite[funpart[w], x, id[cart[V, V]]],
              composite[y, cross[funpart[w], funpart[w]]]],
            p4 -> subclass[composite[cross[composite[funpart[t], u], composite[funpart[w], x]],
               TWIST], composite[cross[v, y], TWIST, cross[cross[funpart[t], funpart[w]],
                cross[funpart[t], funpart[w]]]]]}]] // Reverse

Out[23]= or[not[functor[funpart[t], composite[u, id[cart[V, V]]], v]],
          not[functor[funpart[w], composite[x, id[cart[V, V]]], y]],
          subclass[composite[cross[composite[funpart[t], u], composite[funpart[w], x]],
            TWIST], composite[cross[v, y], TWIST,
            cross[cross[funpart[t], funpart[w]], cross[funpart[t], funpart[w]]]]] == True

In[24]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Theorem.

```
In[25]:= Map[implies[and[functor[funpart[t], composite[u, id[cart[V, V]]], v],
              functor[funpart[w], composite[x, id[cart[V, V]]], y]], #] &,
            (functor[p, q, r] // AssertTest) /. {p → cross[funpart[t], funpart[w]],
             q → direct[composite[u, id[cart[V, V]]], composite[x, id[cart[V, V]]]],
             r -> direct[v, y]}] // MapNotNot

Out[25]= or[functor[cross[funpart[t], funpart[w]],
           composite[cross[u, x], TWIST], composite[cross[v, y], TWIST]],
          not[functor[funpart[t], composite[u, id[cart[V, V]]], v]],
          not[functor[funpart[w], composite[x, id[cart[V, V]]], y]]] == True

In[26]:= (% /. {t → t_, u → u_, v → v_, w → w_, x → x_, y → y_}) /. Equal→ SetDelayed
```

The wrappers can now be removed.

Main Theorem.

```
In[27]:= SubstTest[implies, and[equal[t, funpart[p]], equal[w, funpart[q]],
            equal[u, composite[r, id[cart[V, V]]]], equal[x, composite[s, id[cart[V, V]]]]],
           or[functor[cross[t, w], composite[cross[u, x], TWIST],
            composite[cross[v, y], TWIST]], not[functor[t, u, v]], not[functor[w, x, y]]],
           {p → t, q → w, r → u, s → x}] // Reverse // MapNotNot

Out[27]= or[functor[cross[t, w], composite[cross[u, x], TWIST], composite[cross[v, y], TWIST]],
          not[functor[t, u, v]], not[functor[w, x, y]], not[subclass[u, cart[cart[V, V], V]]],
          not[subclass[x, cart[cart[V, V], V]]]] == True

In[28]:= or[functor[cross[t_, w_], composite[cross[u_, x_], TWIST],
            composite[cross[v_, y_], TWIST]], not[functor[t_, u_, v_]],
          not[functor[w_, x_, y_]], not[subclass[u_, cart[cart[V, V], V]]],
          not[subclass[x_, cart[cart[V, V], V]]]] := True
```

The hypotheses that **u** and **x** be ternary relations are satisfied if they are categories. The following corollary is obtained by wrapping **u** and **x** with **cat**.

Corollary. If **t** is a functor from **cat[u]** to **v** and if **w** is a functor from **cat[x]** to **y**, then the cross product of **t** and **w** is a functor from the direct product of **cat[u]** and **cat[x]** to the direct product of **v** and **y**.

```
In[29]:= SubstTest[or,
           functor[cross[t, w], composite[cross[r, s], TWIST], composite[cross[v, y], TWIST]],
           not[functor[t, r, v]], not[functor[w, s, y]], not[subclass[r, cart[cart[V, V], V]]],
           not[subclass[s, cart[cart[V, V], V]]], {r → cat[u], s → cat[x]}] // Reverse

Out[29]= or[functor[cross[t, w],
           composite[cross[cat[u], cat[x]], TWIST], composite[cross[v, y], TWIST]],
          not[functor[t, cat[u], v]], not[functor[w, cat[x], y]]] == True

In[30]:= or[functor[cross[t_, w_],
            composite[cross[cat[u_], cat[x_]], TWIST], composite[cross[v_, y_], TWIST]],
          not[functor[t_, cat[u_], v_]], not[functor[w_, cat[x_], y_]]] := True
```