direct products of associative relations

Johan G. F. Belinfante
2003 July 17

```
In[1]:= << goedel52.s55; << tools.m

        :Package Title: goedel52.s55       2003 July 16 at 10:35 p.m.

        It is now:  2003 Jul 18 at 14:39

        Loading Simplification Rules

        TOOLS.M                      Revised 2003 July 8

        weightlimit = 40
```

■ summary

This notebook contains a derivation of the fact that the direct product of associative relations is an associative relation. Recall the definition of associative relation:

```
In[2]:= associative[x] // AssertTest

Out[2]= associative[x] ==
         and[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]],
          subclass[x, cart[cart[V, V], V]]]
```

The direct product of two binary relations **x** and **y** is defined as:

```
In[3]:= direct[x_, y_] := composite[cross[x, y], TWIST]
```

This definition is intended to be used only as a temporary abbreviation, and will not, for the time being, be added to the **GOEDEL** program.

■ normalization for direct products

Normalization of the direct product of two binary relations **x** and **y** is achieved as follows:

```
In[4]:= direct[x, y] // VSTriNormality // Reverse

Out[4]= intersection[composite[inverse[FIRST], x, cross[FIRST, FIRST]],
          composite[inverse[SECOND], y, cross[SECOND, SECOND]]] == composite[cross[x, y], TWIST]

In[5]:= intersection[composite[inverse[FIRST], x_, cross[FIRST, FIRST]],
          composite[inverse[SECOND], y_, cross[SECOND, SECOND]]] := composite[cross[x, y], TWIST]
```

With this rewrite rule in place, a transparent description of the direct product can be given:

```
In[6]:= class[pair[pair[pair[u1, u2], pair[v1, v2]], pair[w1, w2]],
            and[member[pair[pair[u1, v1], w1], x], member[pair[pair[u2, v2], w2], y]]]
Out[6]= composite[cross[x, y], TWIST]
```

■ some TWIST simplifications

```
In[7]:= Assoc[TWIST, cross[Id, id[cart[V, V]]], cross[x, Id]]
Out[7]= composite[TWIST, cross[x, id[cart[V, V]]]] == composite[TWIST, cross[x, Id]]

In[8]:= composite[TWIST, cross[x_, id[cart[V, V]]]] := composite[TWIST, cross[x, Id]]

In[9]:= Assoc[TWIST, cross[id[cart[V, V]], Id], cross[Id, x]]
Out[9]= composite[TWIST, cross[id[cart[V, V]], x]] == composite[TWIST, cross[Id, x]]

In[10]:= composite[TWIST, cross[id[cart[V, V]], x_]] := composite[TWIST, cross[Id, x]]

In[11]:= composite[cross[x, id[cart[V, V]]], TWIST] // DoubleInverse
Out[11]= composite[cross[x, id[cart[V, V]]], TWIST] == composite[cross[x, Id], TWIST]

In[12]:= composite[cross[x_, id[cart[V, V]]], TWIST] := composite[cross[x, Id], TWIST]

In[13]:= composite[cross[id[cart[V, V]], x], TWIST] // DoubleInverse
Out[13]= composite[cross[id[cart[V, V]], x], TWIST] == composite[cross[Id, x], TWIST]

In[14]:= composite[cross[id[cart[V, V]], x_], TWIST] := composite[cross[Id, x], TWIST]
```

■ LEFT and RIGHT formulas

```
In[15]:= Assoc[TWIST, id[cart[cart[V, V], V]], LEFT[x]] // Reverse
Out[15]= composite[TWIST, LEFT[x]] == cross[LEFT[first[x]], LEFT[second[x]]]

In[16]:= composite[TWIST, LEFT[x_]] := cross[LEFT[first[x]], LEFT[second[x]]]

In[17]:= Assoc[TWIST, id[cart[V, cart[V, V]]], RIGHT[x]] // Reverse
Out[17]= composite[TWIST, RIGHT[x]] == cross[RIGHT[first[x]], RIGHT[second[x]]]

In[18]:= composite[TWIST, RIGHT[x_]] := cross[RIGHT[first[x]], RIGHT[second[x]]]
```

■ main idea

The main idea is to first show that associativity of **x** and **y** implies the following consequence of associativity of the direct product:

```
In[19]:= Map[not, SubstTest[and, implies[p1, p3], implies[p2, p4], implies[and[p3, p4], p5],
             not[implies[and[p1, p2], p5]],
           {p1 -> equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]],
             p2 -> equal[composite[y, cross[y, Id]], composite[y, cross[Id, y], ASSOC]],
             p3 -> commute[composite[x, LEFT[s]], composite[x, RIGHT[t]]],
             p4 -> commute[composite[y, LEFT[u]], composite[y, RIGHT[v]]],
             p5 -> commute[composite[direct[x, y], LEFT[PAIR[s, u]]],
            composite[direct[x, y], RIGHT[PAIR[t, v]]]]}]] /.
         {s -> first[w], t -> first[z], u -> second[w], v -> second[z]}

Out[19]= or[equal[cross[composite[x, LEFT[first[w]], x, RIGHT[first[z]]],
             composite[y, LEFT[second[w]], y, RIGHT[second[z]]]],
           cross[composite[x, RIGHT[first[z]], x, LEFT[first[w]]],
             composite[y, RIGHT[second[z]], y, LEFT[second[w]]]]],
          not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]],
          not[equal[composite[y, cross[y, Id]], composite[y, cross[Id, y], ASSOC]]]] == True
```

This is added as a temporary rewrite rule.

```
In[20]:= (% /. {w -> w_, x -> x_, y -> y_, z -> z_}) /. Equal -> SetDelayed
```

The remaining problem is to eliminate the variables **w** and **z**. This will be accomplished by using simulated asserted quantifiers. The universal quantifier is defined in terms of negation and the existential quantifier. The negation seems to produces problems when used in conjunction with **SubstTest**, which is avoided by replacing the construction **assert[forall[w,p[w]]]** with the following logically equivalent simulation **equal[V,class[w,p[w]]**. The simulation introduces no negation.

■ protecting the hypotheses

There still is a problem with negation in that the associativity hypotheses of the proposed theorem are wrapped with **not**. The following temporary rule is needed to assure that the associativity hypotheses survive the action of **class**:

```
In[21]:= not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]] //
           AssertTest // Reverse

Out[21]= or[not[subclass[composite[x, cross[Id, x], ASSOC], composite[x, cross[x, Id]]]], not[
            subclass[composite[x, cross[x, Id], inverse[ASSOC]], composite[x, cross[Id, x]]]],
           not[subclass[image[inverse[x], domain[domain[x]]], cart[V, V]]]] ==
          not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]]

In[22]:= or[not[subclass[composite[x_, cross[Id, x_], ASSOC], composite[x_, cross[x_, Id]]]],
           not[subclass[composite[x_, cross[x_, Id], inverse[ASSOC]],
             composite[x_, cross[Id, x_]]]],
           not[subclass[image[inverse[x_], domain[domain[x_]]], cart[V, V]]]] :=
          not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]]
```

■ removing the variables w and z

The simulated asserted universal quantification is very time consuming, so it is essential to turn off the **simplify** flag.

```
In[23]:= simplify = False;

In[24]:= SubstTest[equal, rotate[u], rotate[v], {u -> z, v -> composite[x, cross[y, Id]]}]

Out[24]= equal[composite[rotate[x], cross[Id, y]], rotate[z]] ==
          equal[composite[x, cross[y, Id]], composite[z, id[cart[V, V]]]]
```

```
In[25]:= equal[composite[rotate[x_], cross[Id, y_]], rotate[z_]] :=
          equal[composite[x, cross[y, Id]], composite[z, id[cart[V, V]]]]

In[26]:=     SubstTest[equal, composite[u, RIGHT[w], SWAP], composite[v, RIGHT[w], SWAP],
             {u -> composite[direct[x, y], cross[direct[x, y], Id]],
                 v -> composite[direct[x, y], cross[Id, direct[x, y]], ASSOC]}]

Out[26]= equal[composite[cross[composite[x, RIGHT[first[w]]], x, SWAP],
             composite[y, RIGHT[second[w]], y, SWAP]], TWIST],
           composite[cross[composite[x, SWAP], composite[y, SWAP]], TWIST,
            cross[cross[composite[x, RIGHT[first[w]]], composite[y, RIGHT[second[w]]]], Id]]] ==
          equal[composite[cross[composite[x, RIGHT[first[w]], x],
             composite[y, RIGHT[second[w]], y]], TWIST], composite[cross[x, y], TWIST,
            cross[Id, cross[composite[x, RIGHT[first[w]]], composite[y, RIGHT[second[w]]]]]]]

In[27]:= equal[composite[cross[composite[x_, RIGHT[first[w_]], x_, SWAP],
             composite[y_, RIGHT[second[w_]], y_, SWAP]], TWIST],
           composite[cross[composite[x_, SWAP], composite[y_, SWAP]], TWIST, cross[
             cross[composite[x_, RIGHT[first[w_]]], composite[y_, RIGHT[second[w_]]]], Id]]] :=
          equal[composite[cross[composite[x, RIGHT[first[w]], x],
             composite[y, RIGHT[second[w]], y]], TWIST], composite[cross[x, y], TWIST,
            cross[Id, cross[composite[x, RIGHT[first[w]]], composite[y, RIGHT[second[w]]]]]]]
```

The variable **z** is removed now:

```
In[28]:= Map[equal[V, #] &, SubstTest[class, z,
            implies[and[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]],
                 equal[composite[y, cross[y, Id]], composite[y, cross[Id, y], ASSOC]]],
                 equal[composite[u, LEFT[z]], composite[v, LEFT[z]]]],
              {u -> composite[direct[x, y], cross[direct[x, y], Id], RIGHT[w]],
                 v -> composite[direct[x, y], cross[Id, direct[x, y]], ASSOC,
                     RIGHT[w]]}]] // Reverse

Out[28]= or[equal[composite[cross[composite[x, RIGHT[first[w]], x],
              composite[y, RIGHT[second[w]], y]], TWIST], composite[cross[x, y], TWIST,
             cross[Id, cross[composite[x, RIGHT[first[w]]], composite[y, RIGHT[second[w]]]]]],
           not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]],
           not[equal[composite[y, cross[y, Id]], composite[y, cross[Id, y], ASSOC]]]] == True
```

This is added as a temporary rewrite rule.

```
In[29]:= (% /. {x -> x_, y -> y_, w -> w_}) /. Equal -> SetDelayed
```

Next the variable **w** is removed:

```
In[30]:= Map[equal[V, #] &, SubstTest[class, w, or[equal[composite[u, RIGHT[w]],
                 composite[v, RIGHT[w]]],
                 not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]],
              not[equal[composite[y, cross[y, Id]], composite[y, cross[Id, y], ASSOC]]]],
              {u -> composite[direct[x, y], cross[direct[x, y], Id]], v ->
               composite[direct[x, y], cross[id[cart[V, V]], direct[x, y]], ASSOC]}]] // Reverse

Out[30]= or[equal[composite[cross[x, y], TWIST, cross[composite[cross[x, y], TWIST], Id]],
             composite[cross[x, y], TWIST, cross[Id, composite[cross[x, y], TWIST]], ASSOC]],
           not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]],
           not[equal[composite[y, cross[y, Id]], composite[y, cross[Id, y], ASSOC]]]] == True
```

This is added as a temporary rewrite rule.

```
In[31]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

■ final steps

The following fact is needed to rewrite the results obtained in terms of the predicate **associative**.

```
In[32]:= Map[
          implies[equal[composite[cross[x, y], TWIST, cross[composite[cross[x, y], TWIST], Id]],
             composite[cross[x, y], TWIST, cross[Id, composite[cross[x, y], TWIST]], ASSOC]],
           #] &, ((associative[z] // AssertTest) /. z -> direct[x, y])]

Out[32]= or[associative[composite[cross[x, y], TWIST]],
           not[equal[composite[cross[x, y], TWIST, cross[composite[cross[x, y], TWIST], Id]],
             composite[cross[x, y], TWIST,
              cross[Id, composite[cross[x, y], TWIST]], ASSOC]]]] == True
```

This is added as a temporary rewrite rule.

```
In[33]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

This is the final result:

```
In[34]:= Map[not, SubstTest[and, implies[p1, p3],
           implies[p2, p4], implies[and[p3, p4], p5], implies[p5, p6],
                 not[implies[and[p1, p2], p6]],
             {p1 -> associative[x], p2 -> associative[y],
                 p3 -> equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]],
                 p4 -> equal[composite[y, cross[y, Id]], composite[y, cross[Id, y], ASSOC]],
                 p5 ->
             equal[composite[cross[x, y], TWIST, cross[composite[cross[x, y], TWIST], Id]],
              composite[cross[x, y], TWIST, cross[Id, composite[cross[x, y], TWIST]], ASSOC]],
                 p6 -> associative[direct[x, y]]}]]

Out[34]= or[associative[composite[cross[x, y], TWIST]],
           not[associative[x]], not[associative[y]]] == True

In[35]:= or[associative[composite[cross[x_, y_], TWIST]],
           not[associative[x_]], not[associative[y_]]] := True
```