TWIST formula for associative commutative relations

Johan G. F. Belinfante
2003 August 22

```
In[1]:= << goedel52.s80; << tools.m

        :Package Title: goedel52.s80       2003 August 21 at 5:15 p.m.

        It is now:  2003 Aug 22 at 12:35

        Loading Simplification Rules

        TOOLS.M                       Revised 2003 August 9

        weightlimit = 40
```

summary

It is shown that if **x** is an associative, commutative relation, then **composite[x, cross[x, x]]** is **TWIST**−invariant.

lemmas

The first lemma is needed to connect the function **TWIST** and the functions **ASSOC** and **SWAP**.

```
In[2]:= composite[ASSOC, cross[composite[inverse[ROT], cross[SWAP, Id]], Id], inverse[ASSOC]] //
         VSTriNormality

Out[2]= composite[ASSOC,
          cross[composite[inverse[ROT], cross[SWAP, Id]], Id], inverse[ASSOC]] == TWIST

In[3]:= composite[ASSOC,
          cross[composite[inverse[ROT], cross[SWAP, Id]], Id], inverse[ASSOC]] := TWIST

In[4]:= Assoc[composite[ASSOC, cross[inverse[ASSOC], Id], cross[cross[Id, SWAP], Id]],
         composite[cross[ASSOC, Id], inverse[ASSOC]],
         composite[ASSOC, cross[inverse[ASSOC], Id]]]

Out[4]= composite[ASSOC, cross[composite[inverse[ASSOC], cross[Id, SWAP]], Id]] ==
         composite[TWIST, ASSOC, cross[inverse[ASSOC], Id]]

In[5]:= composite[ASSOC, cross[composite[inverse[ASSOC], cross[Id, SWAP]], Id]] :=
         composite[TWIST, ASSOC, cross[inverse[ASSOC], Id]]
```

The second lemma is a general property of associative relations.

```
In[6]:= SubstTest[implies, equal[u, v], equal[composite[u, w], composite[v, w]],
           {u -> composite[x, cross[x, Id]],
          v -> composite[x, cross[Id, x], ASSOC], w -> inverse[ASSOC]}]

Out[6]= or[equal[composite[x, cross[Id, composite[x, id[cart[V, V]]]]],
            composite[x, cross[x, Id], inverse[ASSOC]]],
          not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]]] == True
```

```
In[7]:= (% /. x -> x_) /. Equal -> SetDelayed
```

```
In[8]:= SubstTest[implies, equal[x, y],
         equal[cross[Id, x], cross[Id, y]], y -> composite[x, id[cart[V, V]]]]
```

```
Out[8]= or[equal[cross[Id, x], cross[Id, composite[x, id[cart[V, V]]]]],
          not[subclass[x, cart[cart[V, V], V]]]] == True
```

```
In[9]:= (% /. x -> x_) /. Equal -> SetDelayed
```

```
In[10]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[p2, p4],
           implies[p3, p5], implies[p4, p6], implies[and[p5, p6], p7], not[implies[p1, p7]],
             {p1 -> associative[x], p2 -> subclass[x, cart[cart[V, V], V]],
            p3 -> equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]],
                 p4 -> equal[cross[Id, x], cross[Id, composite[x, id[cart[V, V]]]]],
                 p5 -> equal[composite[x, cross[Id, composite[x, id[cart[V, V]]]]],
               composite[x, cross[x, Id], inverse[ASSOC]]],
                 p6 -> equal[composite[x, cross[Id, composite[x, id[cart[V, V]]]]],
               composite[x, cross[Id, x]]],
                 p7 -> equal[composite[x, cross[Id, composite[x, Id]]],
               composite[x, cross[x, Id], inverse[ASSOC]]]}]]
```

```
Out[10]= or[equal[composite[x, cross[Id, x]], composite[x, cross[x, Id], inverse[ASSOC]]],
           not[associative[x]]] == True
```

```
In[11]:= or[equal[composite[x_, cross[Id, x_]], composite[x_, cross[x_, Id], inverse[ASSOC]]],
           not[associative[x_]]] := True
```

---

derivation

```
In[12]:= SubstTest[implies, equal[u, v], equal[composite[u, w], composite[v, w]],
             {u -> composite[x, cross[x, Id]],
           v -> composite[x, cross[Id, x], ASSOC], w -> cross[cross[x, Id], Id]}]
```

```
Out[12]= or[equal[composite[x, cross[composite[x, cross[x, Id]], Id]],
             composite[x, cross[x, x], ASSOC]],
           not[equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]]]] == True
```

```
In[13]:= (% /. x -> x_) /. Equal -> SetDelayed
```

```
In[14]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
                 {p1 -> associative[x],
              p2 -> equal[composite[x, cross[x, Id]], composite[x, cross[Id, x], ASSOC]],
                      p3 -> equal[composite[x, cross[composite[x, cross[x, Id]], Id]],
               composite[x, cross[x, x], ASSOC]]}]]
```

```
Out[14]= or[equal[composite[x, cross[composite[x, cross[x, Id]], Id]],
             composite[x, cross[x, x], ASSOC]], not[associative[x]]] == True
```

```
In[15]:= (% /. x -> x_) /. Equal -> SetDelayed
```

```
In[16]:= SubstTest[implies, equal[u, v], equal[composite[u, w], composite[v, w]],
             {u -> composite[x, cross[composite[x, cross[x, Id]], Id]],
           v -> composite[x, cross[x, x], ASSOC], w -> cross[inverse[ASSOC], Id]}]
```

```
Out[16]= or[equal[composite[x, cross[composite[x, cross[x, Id], inverse[ASSOC]], Id]],
             composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]]],
           not[equal[composite[x, cross[composite[x, cross[x, Id]], Id]],
             composite[x, cross[x, x], ASSOC]]]] == True
```

```
In[17]:= (% /. x -> x_) /. Equal -> SetDelayed
```

The story thus far:

```
In[18]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], implies[p1, p4],
           implies[p4, p5], implies[p5, p6], implies[and[p3, p6], p7], not[implies[p1, p7]],
                {p1 -> associative[x], p2 -> equal[composite[x,
             cross[composite[x, cross[x, Id]], Id]], composite[x, cross[x, x], ASSOC]],
               p3 -> equal[composite[x, cross[composite[x, cross[x, Id], inverse[ASSOC]], Id]],
            composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]]],
           p4 -> equal[composite[x, cross[x, Id], inverse[ASSOC]], composite[x, cross[Id, x]]],
               p5 -> equal[cross[composite[x, cross[x, Id], inverse[ASSOC]], Id],
            cross[composite[x, cross[Id, x]], Id]],
               p6 -> equal[composite[x, cross[composite[x,
               cross[x, Id], inverse[ASSOC]], Id]],
                       composite[x, cross[composite[x, cross[Id, x]], Id]]],
               p7 -> equal[composite[x, cross[composite[x, cross[Id, x]], Id]],
            composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]]]}]]

Out[18]= or[equal[composite[x, cross[composite[x, cross[Id, x]], Id]], composite[x,
            cross[x, x], ASSOC, cross[inverse[ASSOC], Id]]], not[associative[x]]] == True

In[19]:= (% /. x -> x_) /. Equal -> SetDelayed

In[20]:= SubstTest[implies, equal[u, v], equal[composite[u, w], composite[v, w]],
              {u -> composite[x, cross[composite[x, cross[Id, x]], Id]],
            v -> composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]],
                  w -> cross[cross[Id, SWAP], Id]}]

Out[20]= or[equal[composite[x, cross[composite[x, cross[Id, composite[x, SWAP]]], Id]],
             composite[x, cross[x, x], TWIST, ASSOC, cross[inverse[ASSOC], Id]]],
           not[equal[composite[x, cross[composite[x, cross[Id, x]], Id]],
             composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]]]]] == True

In[21]:= (% /. x -> x_) /. Equal -> SetDelayed

In[22]:= Map[not, SubstTest[and, implies[p1, p3], implies[p3, p4],
           implies[p2, p5], implies[p5, p6], implies[p6, p7], implies[p7, p8],
           implies[and[p4, p8], p9], not[implies[and[p1, p2], p9]],
             {p1 -> associative[x], p2 -> equal[x, flip[x]],
            p3 -> equal[composite[x, cross[composite[x, cross[Id, x]], Id]],
              composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]]],
            p4 -> equal[composite[x, cross[composite[x, cross[Id, composite[x, SWAP]]], Id]],
              composite[x, cross[x, x], TWIST, ASSOC, cross[inverse[ASSOC], Id]]],
               p5 -> equal[cross[Id, composite[x, SWAP]], cross[Id, x]],
               p6 -> equal[composite[x, cross[Id, composite[x, SWAP]]],
                   composite[x, cross[Id, x]]],
             p7 -> equal[cross[composite[x, cross[Id, composite[x, SWAP]]], Id],
                       cross[composite[x, cross[Id, x]], Id]],
               p8 ->
             equal[composite[x, cross[composite[x, cross[Id, composite[x, SWAP]]], Id]],
                       composite[x, cross[composite[x, cross[Id, x]], Id]]],
               p9 -> equal[composite[x, cross[composite[x, cross[Id, x]], Id]],
              composite[x, cross[x, x], TWIST, ASSOC, cross[inverse[ASSOC], Id]]]}]]

Out[22]= or[equal[composite[x, cross[composite[x, cross[Id, x]], Id]],
             composite[x, cross[x, x], TWIST, ASSOC, cross[inverse[ASSOC], Id]]],
           not[associative[x]], not[equal[x, composite[x, SWAP]]]] == True

In[23]:= (% /. x -> x_) /. Equal -> SetDelayed
```

```
In[24]:= SubstTest[implies, equal[u, v], equal[composite[u, w], composite[v, w]],
           {u -> composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]],
          v -> composite[x, cross[x, x], TWIST, ASSOC, cross[inverse[ASSOC], Id]],
               w -> composite[cross[ASSOC, Id], inverse[ASSOC]]}]

Out[24]= or[equal[
            composite[x, cross[composite[x, id[cart[V, V]]], composite[x, id[cart[V, V]]]]],
            composite[x, cross[x, x], TWIST]],
          not[equal[composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]],
            composite[x, cross[x, x], TWIST, ASSOC, cross[inverse[ASSOC], Id]]]]] == True

In[25]:= (% /. x -> x_) /. Equal -> SetDelayed

In[26]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p1, p4],
           implies[and[p3, p4], p5], implies[p5, p6], not[implies[and[p1, p2], p6]],
             {p1 -> associative[x], p2 -> equal[x, flip[x]],
            p3 -> equal[composite[x, cross[composite[x, cross[Id, x]], Id]],
              composite[x, cross[x, x], TWIST, ASSOC, cross[inverse[ASSOC], Id]]],
             p4 -> equal[composite[x, cross[composite[x, cross[Id, x]], Id]],
              composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]]],
            p5 → equal[composite[x, cross[x, x], ASSOC, cross[inverse[ASSOC], Id]],
              composite[x, cross[x, x], TWIST, ASSOC, cross[inverse[ASSOC], Id]]],
                 p6 -> equal[composite[x, cross[composite[x, id[cart[V, V]]],
                composite[x, id[cart[V, V]]]]], composite[x, cross[x, x], TWIST]]}]]

Out[26]= or[equal[
            composite[x, cross[composite[x, id[cart[V, V]]], composite[x, id[cart[V, V]]]]],
            composite[x, cross[x, x], TWIST]],
          not[associative[x]], not[equal[x, composite[x, SWAP]]]] == True
```

All that remains is to clean this up a bit, removing the unnecessary identity factors.

```
In[27]:= (% /. x -> x_) /. Equal -> SetDelayed

In[28]:= SubstTest[implies, equal[x, y],
           equal[cross[x, x], cross[y, y]], y → composite[x, id[cart[V, V]]]]

Out[28]= or[equal[composite[Id, x], composite[x, id[cart[V, V]]]],
            not[subclass[x, cart[cart[V, V], V]]]] == True

In[29]:= (% /. x -> x_) /. Equal -> SetDelayed

In[30]:= SubstTest[implies, equal[u, v],
           equal[composite[x, u], composite[x, v]], {u → cross[x, x],
            v → cross[composite[x, id[cart[V, V]]], composite[x, id[cart[V, V]]]]}]

Out[30]= or[equal[composite[x, cross[x, x]],
            composite[x, cross[composite[x, id[cart[V, V]]], composite[x, id[cart[V, V]]]]]],
          not[equal[composite[Id, x], composite[x, id[cart[V, V]]]]]] == True

In[31]:= (% /. x -> x_) /. Equal -> SetDelayed

In[32]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
           {p1 -> subclass[x, cart[cart[V, V], V]],
            p2 -> equal[composite[Id, x], composite[x, id[cart[V, V]]]],
            p3 -> equal[composite[x, cross[x, x]], composite[x,
               cross[composite[x, id[cart[V, V]]], composite[x, id[cart[V, V]]]]]]}]]

Out[32]= or[equal[composite[x, cross[x, x]],
            composite[x, cross[composite[x, id[cart[V, V]]], composite[x, id[cart[V, V]]]]]],
          not[subclass[x, cart[cart[V, V], V]]]] == True

In[33]:= (% /. x -> x_) /. Equal -> SetDelayed
```

```
In[34]:= Map[not, SubstTest[and, implies[and [p1, p2], p4], implies[p1, p3],
           implies[p3, p5], implies[and[p4, p5], p6], not[implies[and[p1, p2], p6]],
             {p1 -> associative[x], p2 -> equal[x, flip[x]],
            p3 → subclass[x, cart[cart[V, V], V]], p4 -> equal[composite[x, cross[x, x], TWIST],
              composite[x, cross[composite[x, id[cart[V, V]]], composite[x, id[cart[V, V]]]]]],
            p5 -> equal[composite[x, cross[x, x]], composite[x,
                cross[composite[x, id[cart[V, V]]], composite[x, id[cart[V, V]]]]],
             p6 -> equal[composite[x, cross[x, x], TWIST], composite[x, cross[x, x]]]}]]

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

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

an example

Integer addition is an example of a commutative, associative relation. The **TWIST** formula in this case amounts to the statement that **(u + v) + (x + y) = (u + x) + (v + y)**.

```
In[36]:= SubstTest[implies, and[associative[x], equal[x, flip[x]]],
          equal[composite[x, cross[x, x], TWIST], composite[x, cross[x, x]]],
            x -> INTADD]

Out[36]= equal[composite[INTADD, cross[INTADD, INTADD]],
           composite[INTADD, cross[INTADD, INTADD], TWIST]] == True

In[37]:= composite[INTADD, cross[INTADD, INTADD], TWIST] :=
          composite[INTADD, cross[INTADD, INTADD]]
```

From this one can deduce a property of the set **image[INTADD, id[Z]]** of even integers.

```
In[38]:= ImageComp[composite[INTADD, cross[INTADD, INTADD]], TWIST, id[cart[Z, Z]]]

Out[38]= image[INTADD, id[Z]] == image[INTADD, cart[image[INTADD, id[Z]], image[INTADD, id[Z]]]]
```

This formula says that the set of even integers is closed under addition. A more interesting application is the proof that the sum of two endomorphisms is an endormorphism. This will be discussed in a separate notebook.

comments

The commutativity assumption can not be omitted. For example, the **TWIST** formula does not hold for the associative relation **COMPOSE**. Nonetheless, there exist some non−commutative associative relations that satisfy the **TWIST** formula. For example, the relation **FIRST** is associative, but not commutative, yet it satisfies the formula:

```
In[39]:= equal[composite[FIRST, cross[FIRST, FIRST], TWIST],
           composite[FIRST, cross[FIRST, FIRST]]] // assert

Out[39]= True
```