# functors and inversion

Johan G. F. Belinfante
2011 December 30

```
In[1]:= SetDirectory["l:"]; << goedel.11dec29a
```

```
:Package Title: goedel.11dec29a              2011 December 29 at 9:30 a.m.

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

It is now:  2011 Dec 30 at 15:58

Loading Simplification Rules

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

weightlimit = 40

Loading completed.

It is now:  2011 Dec 30 at 16:10
```

## summary

A functor **t** from a category **x** to a category **y** in general need not intertwine with the inversion functions for **x** and **y**. The inverse of the transform of a morphism of **x** under **t** need not be the transform of its inverse. An intertwining equation does hold for the special case of functors from groupoids to categories.

## a counterexample

It is perhaps best to start with a simple example of a functor that does not intertwine with inversion. The function **PLUS** is a functor from the monoid **NATADD** to the group **INTADD**.

```
In[3]:= functor[PLUS, NATADD, INTADD] // AssertTest
```

```
Out[3]= functor[PLUS, NATADD, INTADD] == True
```

```
In[4]:= functor[PLUS, NATADD, INTADD] := True
```

Both **NATADD** and **INTADD** are categories, but **PLUS** does not intertwine with their inversion functions:

```
In[5]:= (or[equal[composite[t, inv[x]], composite[inv[y], t]], not[functor[t, x, y]]] /.
           {t → PLUS, x → NATADD, y → INTADD}) // assert
```

```
Out[5]= False
```

The morphisms for the categories **NATADD** and **INTADD** are the set $\omega$ of natural numbers and the set **Z** of integers, respectively. The inversion functions for these categories are: **id[{0}]** and **id[Z] ∘ INVERSE**, respectively. The morphisms for the categories **NATADD** and **INTADD** are the set $\omega$ of natural numbers and the set **Z** of integers, respectively. For the functor **PLUS**, the inverse of the transform of a natural number is usually not the transform of the inverse. For example, the transform under **PLUS** of the natural number **1 = set[0] ∈ ω** is the integer **+1 = id[ω] ∘ SUCC ∈ Z**. Its inverse is the integer **–1 = inverse[SUCC] ∘ id[ω] ∈ Z**. On the other hand, the transform of the inverse in this case does not even exist since the natural number **1** does not have an (additive) inverse.

## monotonicity results

Although an interwining equation fails to hold for the general case of a functor from one category to another, a weaker result does hold. In this section two monotonicity inclusions will be derived. To take advantage of all automatic simplifications that may be available, the variable **t** will be wrapped with **funpart** and the variables **x** and **y** will be wrapped with **cat** throughout the derivation. These wrappers can always be removed later. To improve readability, these wrappers will be omitted in the informal notes that accompany the derivations.

Recall that by definition, a functor **t** from **x** to **y** satisfies the inclusion **t ∘ x ⊂ y ∘ (t ⊗ t)**. From this one can deduce an inclusion involving the transformation of the one-sided inverse relation **image[inverse[x], ids[x]]**.

Lemma. An inclusion involving transforms of one-sided inversion.

```
In[6]:= SubstTest[implies, subclass[u, v],
          subclass[image[w, u], image[w, v]], {u -> composite[funpart[t], cat[x]],
           v -> composite[cat[y], cross[funpart[t], funpart[t]]],
           w -> composite[FIRST, id[cart[V, ids[cat[y]]]]]}] // Reverse

Out[6]= or[not[subclass[composite[funpart[t], cat[x]],
            composite[cat[y], cross[funpart[t], funpart[t]]]],
          subclass[image[inverse[cat[x]], image[inverse[funpart[t]], ids[cat[y]]]], composite[
            inverse[funpart[t]], image[inverse[cat[y]], ids[cat[y]]], funpart[t]]]] == True

In[7]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (The inclusion involving the cross product in the hypothesis of the preceding lemma is replaced with a functor hypothesis.)

```
In[8]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 → functor[funpart[t], cat[x], cat[y]], p2 → subclass[
              composite[funpart[t], cat[x]], composite[cat[y], cross[funpart[t], funpart[t]]]],
            p3 -> subclass[image[inverse[cat[x]], image[inverse[funpart[t]], ids[cat[y]]]],
              composite[inverse[funpart[t]],
               image[inverse[cat[y]], ids[cat[y]]], funpart[t]]]}]] // Reverse

Out[8]= or[not[functor[funpart[t], cat[x], cat[y]]],
          subclass[image[inverse[cat[x]], image[inverse[funpart[t]], ids[cat[y]]]], composite[
            inverse[funpart[t]], image[inverse[cat[y]], ids[cat[y]]], funpart[t]]]] == True

In[9]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma.

```
In[10]:= Map[implies[#,
            subclass[inv[cat[x]], composite[inverse[funpart[t]], inv[cat[y]], funpart[t]]]] &,
          SubstTest[subclass, core[t, u], core[t, v],
           {t → SYM, u -> image[inverse[cat[x]], ids[cat[x]]], v ->
             composite[inverse[funpart[t]], image[inverse[cat[y]], ids[cat[y]]], funpart[t]]}]]

Out[10]= or[not[subclass[inv[cat[x]], composite[inverse[funpart[t]],
             image[inverse[cat[y]], ids[cat[y]]], funpart[t]]]], subclass[inv[cat[x]],
           composite[inverse[funpart[t]], inv[cat[y]], funpart[t]]]] == True

In[11]:= (% /. {t → t_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Lemma.

```
In[36]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[p1, p4],
            implies[p4, p5], (*implies[and[p2,p3,p5],p6],*) not[implies[p1, p6]],
            {p1 -> functor[t, x, y], p2 → FUNCTION[t], p3 → subclass[image[t, ids[x]], ids[y]],
             p4 → equal[domain[t], range[x]], p5 → subclass[ids[x], domain[t]],
             p6 -> subclass[ids[x], image[inverse[t], ids[y]]]}]] // Reverse

Out[36]= or[not[functor[t, x, y]], subclass[ids[x], image[inverse[t], ids[y]]]] == True

In[38]:= or[not[functor[t_, x_, y_]], subclass[ids[x_], image[inverse[t_], ids[y_]]]] := True
```

Lemma. An (awkward) form of monotonicity for inversion.

```
In[41]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
            (*implies[and[p2,p3],p4],*) implies[p4, p5], implies[p5, p6] ,
            not[implies[p1, p6]], {p1 → functor[funpart[t], cat[x], cat[y]],
             p2 → contains[composite[inverse[funpart[t]], image[inverse[cat[y]], ids[cat[y]]],
                funpart[t]], image[inverse[cat[x]], image[inverse[funpart[t]], ids[cat[y]]]]],
             p3 → subclass[ids[cat[x]], image[inverse[funpart[t]], ids[cat[y]]]],
             p4 → subclass[image[inverse[cat[x]], ids[cat[x]]], composite[
                inverse[funpart[t]], image[inverse[cat[y]], ids[cat[y]]], funpart[t]]],
             p5 -> subclass[inv[cat[x]], composite[inverse[funpart[t]],
                image[inverse[cat[y]], ids[cat[y]]], funpart[t]]], p6 -> subclass[inv[cat[x]],
               composite[inverse[funpart[t]], inv[cat[y]], funpart[t]]]}]] // Reverse

Out[41]= or[not[functor[funpart[t], cat[x], cat[y]]], subclass[inv[cat[x]],
           composite[inverse[funpart[t]], inv[cat[y]], funpart[t]]]] == True

In[42]:= (% /. {t → t_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Lemma. Simplification rule to simplify wrapper removal.

```
In[44]:= equiv[and[FUNCTION[x], functor[x, y, z]], functor[x, y, z]]

Out[44]= True

In[45]:= and[FUNCTION[x_], functor[x_, y_, z_]] := functor[x, y, z]
```

First Theorem. If **t** is a functor from a category **x** to a category **y**, then **inv[x] ⊂ inverse[t] ∘ inv[y] ∘ t**.

```
In[46]:= SubstTest[implies, and[equal[t, funpart[w]], equal[x, cat[u]], equal[y, cat[v]]],
           or[not[functor[t, x, y]], subclass[inv[x], composite[inverse[t], inv[y], t]]],
           {w → t, u → x, v → y}] // MapNotNot // Reverse

Out[46]= or[not[category[x]], not[category[y]], not[functor[t, x, y]],
           subclass[inv[x], composite[inverse[t], inv[y], t]]] == True

In[47]:= or[not[category[x_]], not[category[y_]], not[functor[t_, x_, y_]],
           subclass[inv[x_], composite[inverse[t_], inv[y_], t_]]] := True
```

Lemma. A more convenient but slightly weaker form of monotonicity.

```
In[48]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           not[implies[p1, p3]], {p1 -> functor[funpart[t], cat[x], cat[y]], p2 ->
             subclass[inv[cat[x]], composite[inverse[funpart[t]], inv[cat[y]], funpart[t]]],
            p3 -> subclass[composite[funpart[t], inv[cat[x]], inverse[funpart[t]]],
              inv[cat[y]]]}]] // Reverse

Out[48]= or[not[functor[funpart[t], cat[x], cat[y]]], subclass[
           composite[funpart[t], inv[cat[x]], inverse[funpart[t]]], inv[cat[y]]]] == True

In[49]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Second Theorem. Any functor **t** from a category **x** to a category **y** is monotone with respect to inversion.

```
In[50]:= SubstTest[implies, and[equal[t, funpart[w]], equal[x, cat[u]], equal[y, cat[v]]],
           or[not[functor[t, x, y]], subclass[composite[t, inv[x], inverse[t]], inv[y]]],
           {w → t, u → x, v → y}] // MapNotNot // Reverse

Out[50]= or[not[category[x]], not[category[y]], not[functor[t, x, y]],
           subclass[composite[t, inv[x], inverse[t]], inv[y]]] == True

In[51]:= or[not[category[x_]], not[category[y_]], not[functor[t_, x_, y_]],
           subclass[composite[t_, inv[x_], inverse[t_]], inv[y_]]] := True
```

This second theorem can be restated using the constructor **monotone**.

```
In[52]:= implies[and[functor[t, x, y], category[x], category[y]],
           subclass[P[composite[Id, t]], monotone[inv[x], inv[y]]]]

Out[52]= True
```

## an interwining equation

In this section it is shown that a functor **t** from a groupoid **x** to a category **y** satisfies an intertwining equation. A category **x** is a **groupoid** if every morphism is invertible, that is, if **range[x] = domain[inv[x]]**. For simplicity, the **funpart** and **cat** wrappers used in the preceding section will be retained, but one minor problem with doing so should be mentioned. To derive an intertwining equation, it would seem that the first step would be to eliminate the inverse function **inverse[t]** from the monotonicity inclusion to obtain an inclusion involving only functions. When **funpart** wrappers are employed, this is blocked by the following rewrite rule that automatically kicks in.

```
In[53]:= subclass[composite[funpart[t], inv[cat[x]]], composite[inv[cat[y]], funpart[t]]]

Out[53]= and[subclass[composite[funpart[t], inv[cat[x]]], inverse[funpart[t]]], inv[cat[y]]],
          subclass[image[inv[cat[x]], domain[funpart[t]]], domain[funpart[t]]]]
```

Rather than eliminating the **funpart** wrapper, the following lemma will be used as a workaround for this problem. This lemma introduces an equation which holds under two hypotheses.

Lemma. (A subclass of a function is a restriction.)

```
In[54]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]], {u -> composite[funpart[t], inv[cat[x]]],
            v -> composite[inv[cat[y]], funpart[t]]}] // Reverse

Out[54]= or[equal[composite[funpart[t], inv[cat[x]]],
            composite[inv[cat[y]], funpart[t], id[image[inv[cat[x]], domain[funpart[t]]]]]],
           not[subclass[composite[funpart[t], inv[cat[x]]], inverse[funpart[t]]], inv[cat[y]]]],
           not[subclass[image[inv[cat[x]], domain[funpart[t]]], domain[funpart[t]]]] == True

In[55]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

The two hypotheses in the above lemma both follow from the conclusion of the first monotonicity theorem derived in the previous section. For the first hypothesis, this is automatic. For the second hypothesis, the following lemma works.

Lemma. (The domain of the functor **t** is invariant under **inv[x]**.)

```
In[56]:= SubstTest[implies, subclass[v, w],
           subclass[composite[u, v], composite[u, w]], {u → funpart[t], v -> inv[cat[x]], w ->
             composite[inverse[funpart[t]], inv[cat[y]], funpart[t]]}] // Reverse // MapNotNot

Out[56]= or[
           not[subclass[inv[cat[x]], composite[inverse[funpart[t]], inv[cat[y]], funpart[t]]]],
           subclass[image[inv[cat[x]], domain[funpart[t]]], domain[funpart[t]]]] == True

In[57]:= (% /. {t → t_, x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. (An equation that will be simplified momentarily.)

```
In[58]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
           implies[p2, p4], implies[and[p3, p4], p5] , not[implies[p1, p5]],
           {p1 → functor[funpart[t], cat[x], cat[y]], p2 -> subclass[inv[cat[x]],
              composite[inverse[funpart[t]], inv[cat[y]], funpart[t]]], p3 ->
             subclass[composite[funpart[t], inv[cat[x]], inverse[funpart[t]]], inv[cat[y]]],
            p4 -> subclass[image[inv[cat[x]], domain[funpart[t]]], domain[funpart[t]]],
            p5 -> equal[composite[funpart[t], inv[cat[x]]], composite[inv[cat[y]],
               funpart[t], id[image[inv[cat[x]], domain[funpart[t]]]]]]}]] // Reverse

Out[58]= or[equal[composite[funpart[t], inv[cat[x]]],
           composite[inv[cat[y]], funpart[t], id[image[inv[cat[x]], domain[funpart[t]]]]]],
          not[functor[funpart[t], cat[x], cat[y]]] == True

In[59]:= (% /. {t → t_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Lemma. (A simplification of the preceding equation.)

```
In[60]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3], implies[and[p2, p3], p4],
           not[implies[p1, p4]], {p1 → functor[funpart[t], cat[x], cat[y]],
            p2 -> equal[composite[funpart[t], inv[cat[x]]],
              composite[inv[cat[y]], funpart[t], id[image[inv[cat[x]], domain[funpart[t]]]]]],
            p3 → equal[domain[funpart[t]], range[cat[x]]],
            p4 -> equal[composite[funpart[t], inv[cat[x]]],
              composite[inv[cat[y]], funpart[t], id[domain[inv[cat[x]]]]]]}]] // Reverse

Out[60]= or[equal[composite[funpart[t], inv[cat[x]]],
           composite[inv[cat[y]], funpart[t], id[domain[inv[cat[x]]]]]],
          not[functor[funpart[t], cat[x], cat[y]]] == True

In[61]:= (% /. {t → t_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Theorem. The intetwining equation for the case that **x** is a groupoid.

```
In[62]:= Map[not, SubstTest[and, implies[p1, p3], implies[p1, p4], implies[and[p2, p3, p4], p5],
           not[implies[and[p1, p2], p5]], {p1 → functor[funpart[t], cat[x], cat[y]],
            p2 → equal[range[cat[x]], domain[inv[cat[x]]]],
            p3 -> equal[composite[funpart[t], inv[cat[x]]],
              composite[inv[cat[y]], funpart[t], id[domain[inv[cat[x]]]]]],
            p4 → equal[domain[funpart[t]], range[cat[x]]], p5 -> equal[composite[
               funpart[t], inv[cat[x]]], composite[inv[cat[y]], funpart[t]]]}]] // Reverse

Out[62]= or[equal[composite[funpart[t], inv[cat[x]]], composite[inv[cat[y]], funpart[t]]],
          not[equal[domain[inv[cat[x]]], range[cat[x]]]],
          not[functor[funpart[t], cat[x], cat[y]]] == True

In[63]:= (% /. {t → t_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Main Theorem. Functors from groupoids to categories intertwine with the inversion functions. Explicitly, if **t** is a functor from a groupoid **x** to a category **y**, then **t** ∘ **inv[x] = inv[y]** ∘ **t**.

```
In[64]:= SubstTest[implies, and[equal[t, funpart[w]], equal[x, cat[u]], equal[y, cat[v]]],
           implies[and [functor[t, x, y], equal[domain[inv[x]], range[x]]],
            intertwine[t, inv[x], inv[y]]], {w → t, u → x, v → y}] // MapNotNot // Reverse

Out[64]= or[equal[composite[t, inv[x]], composite[inv[y], t]], not[category[x]],
          not[category[y]], not[equal[domain[inv[x]], range[x]]], not[functor[t, x, y]]] == True

In[65]:= or[equal[composite[inv[y_], t_], composite[t_, inv[x_]]],
          not[category[x_]], not[category[y_]],
          not[equal[domain[inv[x_]], range[x_]]], not[functor[t_, x_, y_]]] := True
```

---

## an example

The restriction of any category to the cartesian square of its class of identity morphisms is a subcategory that is a groupoid.

Theorem.

```
In[69]:= functor[id[ids[x]], composite[id[ids[x]], inverse[DUP]], x] // AssertTest

Out[69]= functor[id[ids[x]], composite[id[ids[x]], inverse[DUP]], x] == True

In[70]:= functor[id[ids[x_]], composite[id[ids[x_]], inverse[DUP]], x_] := True
```

Theorem. The intertwining equation holds when **x** is a category.

```
In[81]:= SubstTest[implies, equal[x, cat[t]], intertwine[id[ids[x]],
           inv[composite[id[ids[x]], inverse[DUP]]], inv[x]], t → x] // Reverse

Out[81]= or[equal[composite[inv[x], id[ids[x]]], id[ids[x]]], not[category[x]]] == True

In[82]:= or[equal[composite[inv[x_], id[ids[x_]]], id[ids[x_]]], not[category[x_]]] := True
```