functors from monoids to categories

Johan G. F. Belinfante
2011 December 31

```
In[1]:= SetDirectory["l:"]; << goedel.11dec30a
```

```
:Package Title: goedel.11dec30a              2011 December 30 at 5:50 p.m.

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

It is now:  2011 Dec 31 at 21:56

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 31 at 22:10
```

summary

The defintions of **binary homomorphisms** and **functors** were designed to be used in the theory of binary operations and in category theory, respectively. Monoids, and groups in particular, are both binary operations and categories, so for them both concepts are available. From a technical point of view, the statements **functor[t, x, y]** and **t ∈ binhom[x, y]** both have in common that they imply **t** is a function satisfying **t ∘ x ⊂ y ∘ (t ⊗ t)**. For the functor statement it is required also that **domain[t] = range[x]** and that **t** preserves identity (neutral) elements. For the binary homomorphism statement, on the other hand **t** must be a mapping from **fix[domain[x]]** to **fix[domain[y]]** and the equation **t ∘ x = y ∘ (t ⊗ t)** holds. In this notebook it is shown that any functor from a monoid to a category satisfies this equation.

mapping statements for binhoms

The term **monoid** in the **GOEDEL** program refers to a binary composition law; the underlying set on which it acts is its range. For a monoid, the domain is the cartesian square of the range, and hence **fix[domain[x]] = range[x]**.

Lemma. If **x** is a category, then **fix[domain[x]] ⊂ range[x]**.

```
In[2]:= SubstTest[implies, equal[x, cat[t]], subclass[fix[domain[x]], range[x]], t → x] // Reverse
```

```
Out[2]= or[not[category[x]], subclass[fix[domain[x]], range[x]]] == True
```

```
In[3]:= or[not[category[x_]], subclass[fix[domain[x_]], range[x_]]] := True
```

Theorem. A binary homomorphism from a monoid **x** to a category **y** is a mapping from the range of **x** to the range of **y**.

```
In[4]:= Map[not, SubstTest[and, (*implies[p1,p4],implies[p2,p5],implies[p3,p6],*)
          implies[and[p4, p5, p6], p7], not[implies[and[p1, p2, p3], p7]],
          {p1 → member[t, binhom[x, y]], p2 → member[x, MONOIDS],
           p3 → category[y], p4 → member[t, map[fix[domain[x]], fix[domain[y]]]],
           p5 → equal[fix[domain[x]], range[x]], p6 → subclass[fix[domain[y]], range[y]],
           p7 → member[t, map[range[x], range[y]]]}]] // Reverse

Out[4]= or[member[t, map[range[x], range[y]]], not[category[y]],
          not[member[t, binhom[x, y]]], not[member[x, MONOIDS]]] == True

In[5]:= or[member[t_, map[range[x_], range[y_]]], not[category[y_]],
          not[member[t_, binhom[x_, y_]]], not[member[x_, MONOIDS]]] := True
```

---

functor ⇒ binhom

In this section it is shown that the equation **t ∘ x = y ∘ (t ⊗ t)** holds for any functor from a monoid **x** to a category **y**. One can even drop the requirements that **x** be a set, or even a category.

Theorem. If **domain[x] = range[x] × range[x]**, then any functor **t** from **x** to a category **y** satisfies the equation **t ∘ x = y ∘ (t ⊗ t)** .

```
In[6]:= Map[not, SubstTest[and, (*implies[and[p1,p3],p4],implies[p1,p5],*)
          implies[and[p2, p4, p5], p6], not[implies[and[p1, p2, p3], p6]], {p1 → functor[t, x, y],
           p2 → equal[domain[x], cart[range[x], range[x]]], p3 → category[y],
           p4 → equal[composite[t, x], composite[y, cross[t, t], id[domain[x]]]],
           p5 → equal[domain[t], range[x]],
           p6 -> equal[composite[t, x], composite[y, cross[t, t]]]}]] // Reverse

Out[6]= or[equal[composite[t, x], composite[y, cross[t, t]]], not[category[y]],
          not[equal[cart[range[x], range[x]], domain[x]]], not[functor[t, x, y]]] == True

In[7]:= or[equal[composite[t_, x_], composite[y_, cross[t_, t_]]], not[category[y_]],
          not[equal[cart[range[x_], range[x_]], domain[x_]]], not[functor[t_, x_, y_]]] := True
```

Corollary. If **t** is a functor from a monoid **x** to a category **y**, then **t ∘ x = y ∘ (t ⊗ t)** .

```
In[8]:= Map[not, SubstTest[and, implies[p2, p4], implies[and[p1, p3, p4], p5],
          not[implies[and[p1, p2, p3], p5]], {p1 → functor[t, x, y], p2 → member[x, MONOIDS],
           p3 → category[y], p4 -> equal[domain[x], cart[range[x], range[x]]],
           p5 -> equal[composite[t, x], composite[y, cross[t, t]]]}]] // Reverse

Out[8]= or[equal[composite[t, x], composite[y, cross[t, t]]],
          not[category[y]], not[functor[t, x, y]], not[member[x, MONOIDS]]] == True

In[9]:= or[equal[composite[t_, x_], composite[y_, cross[t_, t_]]],
          not[category[y_]], not[functor[t_, x_, y_]], not[member[x_, MONOIDS]]] := True
```

Corollary. The equation **t ∘ x = y ∘ (t ⊗ t)**. holds for any functor **t** from a category **x** with a single identity morphism to a category **y**.

```
In[11]:= Map[not, SubstTest[and, implies[p2, p4],
           implies[and[p1, p3, p4], p5], not[implies[and[p1, p2, p3], p5]],
           {p1 → functor[t, x, y], p2 → and[category[x], member[ids[x], range[SINGLETON]]],
            p3 → category[y], p4 -> equal[domain[x], cart[range[x], range[x]]],
            p5 -> equal[composite[t, x], composite[y, cross[t, t]]]}]] // Reverse

Out[11]= or[equal[composite[t, x], composite[y, cross[t, t]]], not[category[x]],
           not[category[y]], not[functor[t, x, y]], not[member[ids[x], range[SINGLETON]]]] == True

In[13]:= or[equal[composite[t_, x_], composite[y_, cross[t_, t_]]],
           not[category[x_]], not[category[y_]], not[functor[t_, x_, y_]],
           not[member[ids[x_], range[SINGLETON]]]] := True
```

Theorem. Functors between monoids are binary homomorphisms.

```
In[14]:= Map[not,
           SubstTest[and, (*implies[p1,p4],implies[p1,p5],implies[p1,p6],*) implies[p1, p7],
            (*implies[and[p1,p7],p8],*) implies[and[p4, p5, p6, p8], p9], not[implies[p1, p9]],
            {p1 → and[functor[t, x, y], member[x, MONOIDS], member[y, MONOIDS]],
             p4 -> member[t, map[range[x], range[y]]], p5 → equal[range[x], fix[domain[x]]],
             p6 → equal[range[y], fix[domain[y]]], p7 → category[y],
             p8 -> equal[composite[t, x], composite[y, cross[t, t]]],
             p9 → member[t, binhom[x, y]]}]] // Reverse

Out[14]= or[member[t, binhom[x, y]], not[functor[t, x, y]],
           not[member[x, MONOIDS]], not[member[y, MONOIDS]]] == True

In[15]:= or[member[t_, binhom[x_, y_]], not[functor[t_, x_, y_]],
           not[member[x_, MONOIDS]], not[member[y_, MONOIDS]]] := True
```

Corollary. Functors between groups are binary homomorphisms.

```
In[16]:= Map[not, SubstTest[and, (*implies[p1,p2],implies[p1,p3],*)
            implies[and[p0, p2, p3], p4], not[implies[and[p0, p1], p4]], {p0 -> functor[t, x, y],
             p1 → and[member[x, GROUPS], member[y, GROUPS]], p2 → member[x, MONOIDS],
             p3 → member[y, MONOIDS], p4 -> member[t, binhom[x, y]]}]] // Reverse

Out[16]= or[member[t, binhom[x, y]], not[functor[t, x, y]],
           not[member[x, GROUPS]], not[member[y, GROUPS]]] == True

In[17]:= or[member[t_, binhom[x_, y_]], not[functor[t_, x_, y_]],
           not[member[x_, GROUPS]], not[member[y_, GROUPS]]] := True
```

A binary homomorphism **t** from a monoid **x** to a monoid **y** is said to be **unital** if it preserves neutral elements, that is, if **APPLY[t, e[x]] = e[y]**. (For groups this is automatic.)

Theorem. Functors from one monoid to another are unital.

```
In[18]:= Map[not,
           SubstTest[and, (*implies[p1,p2],implies[p1,p3],implies[p1,p4],implies[p1,p5],*)
            implies[and[p2, p4], p6], implies[and[p3, p5, p6], p7], not[implies[p1, p7]],
            {p1 → and[functor[t, x, y], member[x, MONOIDS], member[y, MONOIDS]],
             p2 → member[e[x], ids[x]], p3 → equal[ids[y], set[e[y]]],
             p4 → subclass[ids[x], image[inverse[t], ids[y]]], p5 → FUNCTION[t],
             p6 → member[e[x], image[inverse[t], ids[y]]], p7 -> equal[APPLY[t, e[x]], e[y]]}]] //
          Reverse

Out[18]= or[equal[APPLY[t, e[x]], e[y]], not[functor[t, x, y]],
           not[member[x, MONOIDS]], not[member[y, MONOIDS]]] == True

In[19]:= or[equal[APPLY[t_, e[x_]], e[y_]], not[functor[t_, x_, y_]],
           not[member[x_, MONOIDS]], not[member[y_, MONOIDS]]] := True
```

---

binhom ⇒ functor

Theorem. A binary homomorphism **t** from a monoid **x** to a monoid **y** is a mapping from **range[x]** to **range[y]]**.

```
In[20]:= Map[not, SubstTest[and, implies[p2, p4],
            implies[and[p1, p3, p4], p5], not[implies[and[p1, p2, p3], p5]],
            {p1 -> member[x, MONOIDS], p2 -> member[y, MONOIDS], p3 -> member[t, binhom[x, y]],
             p4 → category[y], p5 → member[t, map[range[x], range[y]]]}]] // Reverse

Out[20]= or[member[t, map[range[x], range[y]]], not[member[t, binhom[x, y]]],
           not[member[x, MONOIDS]], not[member[y, MONOIDS]]] == True

In[21]:= or[member[t_, map[range[x_], range[y_]]], not[member[t_, binhom[x_, y_]]],
           not[member[x_, MONOIDS]], not[member[y_, MONOIDS]]] := True
```

Corollary. If **t** is a binary homomorphism from a monoid **x** to a monoid **y**, then **domain[t] = range[x]**.

```
In[22]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
            {p1 → and[member[t, binhom[x, y]], member[x, MONOIDS], member[y, MONOIDS]], p2 ->
              member[t, map[range[x], range[y]]], p3 -> equal[domain[t], range[x]]}]] // Reverse

Out[22]= or[equal[domain[t], range[x]], not[member[t, binhom[x, y]]],
           not[member[x, MONOIDS]], not[member[y, MONOIDS]]] == True

In[23]:= or[equal[domain[t_], range[x_]], not[member[t_, binhom[x_, y_]]],
           not[member[x_, MONOIDS]], not[member[y_, MONOIDS]]] := True
```

Lemma.

```
In[24]:= SubstTest[implies, equal[t, funpart[z]],
           subclass[image[t, set[x]], set[APPLY[t, x]]], z → t] // Reverse

Out[24]= or[not[FUNCTION[t]], subclass[image[t, set[x]], set[APPLY[t, x]]]] == True

In[25]:= or[not[FUNCTION[t_]], subclass[image[t_, set[x_]], set[APPLY[t_, x_]]]] := True
```

Lemma.

```
In[26]:= Map[not, SubstTest[and, (*implies[p1,p3],implies[p1,p4],
             implies[p1,p5],*)implies[and[p2, p3, p4, p5], p6], not[implies[and[p1, p2], p6]],
            {p1 → and[member[x, MONOIDS], member[y, MONOIDS], member[t, binhom[x, y]]],
             p2 -> equal[APPLY[t, e[x]], e[y]], p3 → equal[ids[x], set[e[x]]],
             p4 → equal[ids[y], set[e[y]]], p5 → FUNCTION[t],
             p6 -> subclass[image[t, ids[x]], ids[y]]}]] // Reverse

Out[26]= or[not[equal[APPLY[t, e[x]], e[y]]],
           not[member[t, binhom[x, y]]], not[member[x, MONOIDS]],
           not[member[y, MONOIDS]], subclass[image[t, ids[x]], ids[y]]] == True

In[27]:= (% /. {t → t_, x → x_, y → y_}) /. Equal→ SetDelayed
```

Theorem. A unital binary homomorphism **t** from a monoid **x** to a monoid **y** is a functor from **x** to **y**.

```
In[28]:= Map[not, SubstTest[and, (*implies[p1,p2],*) implies[p1, p3],
            (*implies[p3,p4],*) implies[p1, p5], (*implies[p5,p6],implies[p1,p7],*)
            implies[and[p2, p4, p6, p7], p8], not[implies[p1, p8]], {p1 → and[member[x, MONOIDS],
              member[y, MONOIDS], member[t, binhom[x, y]], equal[APPLY[t, e[x]], e[y]]],
             p2 → FUNCTION[t], p3 → equal[composite[t, x], composite[y, cross[t, t]]],
             p4 → subclass[composite[t, x], composite[y, cross[t, t]]],
             p5 → member[t, map[range[x], range[y]]], p6 -> equal[domain[t], range[x]],
             p7 → subclass[image[t, ids[x]], ids[y]], p8 → functor[t, x, y]}]] // Reverse

Out[28]= or[functor[t, x, y], not[equal[APPLY[t, e[x]], e[y]]], not[member[t, binhom[x, y]]],
           not[member[x, MONOIDS]], not[member[y, MONOIDS]]] == True

In[29]:= or[functor[t_, x_, y_],
           not[equal[APPLY[t_, e[x_]], e[y_]]], not[member[t_, binhom[x_, y_]]],
           not[member[x_, MONOIDS]], not[member[y_, MONOIDS]]] := True
```

For binary homomorphisms between groups, the unital condition can be omitted.

Corollary. If **t** is a binary homomorphism from a group **x** to a group **y**, then **t** is a functor from **x** to **y**.

```
In[30]:= Map[not, SubstTest[and, (*implies[p1,p3],implies[p1,p4],implies[and[p1,p2],p5],*)
            implies[and[p2, p3, p4, p5], p6], not[implies[and[p1, p2], p6]],
            {p1 → and[member[x, GROUPS], member[y, GROUPS]],
             p2 -> member[t, binhom[x, y]], p3 → member[x, MONOIDS], p4 → member[y, MONOIDS],
             p5 -> equal[APPLY[t, e[x]], e[y]], p6 → functor[t, x, y]}]] // Reverse

Out[30]= or[functor[t, x, y], not[member[t, binhom[x, y]]],
           not[member[x, GROUPS]], not[member[y, GROUPS]]] == True

In[31]:= or[functor[t_, x_, y_], not[member[t_, binhom[x_, y_]]],
           not[member[x_, GROUPS]], not[member[y_, GROUPS]]] := True
```

Corollary. A conditional rewrite rule.

```
In[32]:= implies[and[member[x, GROUPS], member[y, GROUPS]],
           equiv[functor[t, x, y], member[t, binhom[x, y]]]] // not // not

Out[32]= True

In[33]:= functor[t_, x_, y_] :=
          member[t, binhom[x, y]] /; and[member[x, GROUPS], member[y, GROUPS]]
```