CORE[x] is idempotent

Johan G. F. Belinfante and Tiffany Goble
2003 September 4

```
In[1]:= << goedel52.s87; << tools.m

        :Package Title: goedel52.s87       2003 September 2 at 1:35 p.m.

        It is now:  2003 Sep 5 at 9:0

        Loading Simplification Rules

        TOOLS.M                      Revised 2003 August 9

        weightlimit = 40
```

summary

The function **CORE[x]** is idempotent. This fact is rederived in this notebook, along with some related facts. No conditions are put on the class **x**. The present version of the **GOEDEL** program already has a rewrite rule that expreses the idempotence of **CORE[x]**.

```
In[2]:= composite[CORE[x], CORE[x]]

Out[2]= CORE[x]
```

This rule is now removed, and will later be rederived:

```
In[3]:= composite[CORE[x_], CORE[x_]] =.
```

When **x** is a subclass of **y** or conversely, the functions **CORE[x]** and **CORE[y]** commute, and their composite is **CORE[intersection[x, y]]**. A simple formula for **CORE[union[x,y]]** is derived which holds without any conditions on **x** and **y**. As a byproduct, an interesting formula for the power class of a union emerges.

core[x,y]

A temporary abbreviation is introduced for the **x**−core of **y**.

```
In[4]:= core[x_, y_] := U[intersection[x, P[y]]]
```

In the case that **x** is a topology and **y** is a set, this is what one usually calls the interior of **y**. In the present notebook, neither **x** nor **y** is assumed to be a set, and none of the topology axioms are needed. The following fact will be needed below:

```
In[5]:= subclass[core[x, y], y]

Out[5]= True
```

monotonicity

The monotonicity of **core[x,y]** with respect to its second argument can be derived quickly as follows:

```
In[6]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
         {u → P[y], v → P[z], w → composite[inverse[E], id[x]]}]

Out[6]= or[not[subclass[y, z]],
          subclass[U[intersection[x, P[y]]], U[intersection[x, P[z]]]]] == True
```

This is worth adding as a permanent rewrite rule:

```
In[7]:= or[not[subclass[y_, z_]],
          subclass[U[intersection[x_, P[y_]]], U[intersection[x_, P[z_]]]]] := True
```

Restatement:

```
In[8]:= implies[subclass[y, z], subclass[core[x, y], core[x, z]]]

Out[8]= True
```

No new rule is needed for monotonicity with respect to the first argument.

```
In[9]:= implies[subclass[x, y], subclass[core[x, z], core[y, z]]]

Out[9]= True
```

idempotence

The core of a core involves the construction **P[U[−]]**:

```
In[10]:= core[x, core[x, y]]

Out[10]= U[intersection[x, P[U[intersection[x, P[y]]]]]]
```

The following general inclusion holds for this construction:

```
In[11]:= subclass[z, P[U[z]]]

Out[11]= True
```

The following application of **SubstTest** makes use of this fact:

```
In[12]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → intersection[x, P[y]], v → P[core[x, y]], w → composite[inverse[E], id[x]]}]

Out[12]= subclass[U[intersection[x, P[y]]],
           U[intersection[x, P[U[intersection[x, P[y]]]]]]] == True

In[13]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The reverse inclusion also holds, so an equation holds:

```
In[14]:= SubstTest[and, subclass[u, v], subclass[v, u], {u → core[x, y], v → core[x, core[x, y]]}]

Out[14]= True == equal[U[intersection[x, P[y]]], U[intersection[x, P[U[intersection[x, P[y]]]]]]]

In[15]:= U[intersection[x_, P[U[intersection[x_, P[y_]]]]]] := U[intersection[x, P[y]]]
```

Restatement:

```
In[16]:= equal[core[x, core[x, y]], core[x, y]]

Out[16]= True
```

a corollary

The following corollary makes it unnecessary to retain the result of the preceding section as a permanent rewrite rule.

```
In[17]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u → intersection[x, P[y]], v → intersection[x, P[core[x, y]]]}]

Out[17]= True == equal[intersection[x, P[y]], intersection[x, P[U[intersection[x, P[y]]]]]]

In[18]:= intersection[x_, P[U[intersection[x_, P[y_]]]]] := intersection[x, P[y]]
```

In the setting of topology, the interpretation of this result is that the collection of open subsets of a set is the same as the collection of open subsets of the interior of that set.

functional formulations

The corollary of the preceding section has the following functional counterpart:

```
In[19]:= composite[IMAGE[id[x]], POWER, CORE[x]] // VSNormality

Out[19]= composite[IMAGE[id[x]], POWER, CORE[x]] == composite[IMAGE[id[x]], POWER]

In[20]:= composite[IMAGE[id[x_]], POWER, CORE[x_]] := composite[IMAGE[id[x]], POWER]
```

Application of the associative law yields the idempotence rule that was removed at the beginning of this notebook.

```
In[21]:= Assoc[BIGCUP, composite[IMAGE[id[x]], POWER], CORE[x]] // Reverse

Out[21]= composite[CORE[x], CORE[x]] == CORE[x]
```

The rewrite rule is now restored:

```
In[22]:= composite[CORE[x_], CORE[x_]] := CORE[x]
```

a more general result

The method used to derive the idempotence property can be used to derive a more general result:

```
In[23]:= Assoc[composite[BIGCUP, IMAGE[id[x]]],
           composite[IMAGE[id[y]], POWER], CORE[y]] // Reverse

Out[23]= composite[CORE[intersection[x, y]], CORE[y]] == CORE[intersection[x, y]]

In[24]:= composite[CORE[intersection[x_, y_]], CORE[y_]] := CORE[intersection[x, y]]
```

This result can be reformulated. To do so, we need three lemmas:

```
In[25]:= SubstTest[implies, equal[x, z], equal[CORE[x], CORE[z]], z -> intersection[x, y]]

Out[25]= or[equal[Uclosure[x], Uclosure[intersection[x, y]]], not[subclass[x, y]]] == True

In[26]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

In[27]:= SubstTest[implies, equal[u, v], equal[composite[u, w], composite[v, w]],
             {u -> CORE[x], v -> CORE[intersection[x, y]], w -> CORE[y]}]

Out[27]= or[equal[composite[CORE[x], CORE[y]], CORE[intersection[x, y]]],
           not[equal[Uclosure[x], Uclosure[intersection[x, y]]]]] == True

In[28]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

In[29]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
             {u -> CORE[x], v -> CORE[intersection[x, y]], w -> composite[CORE[x], CORE[y]]}]

Out[29]= or[equal[composite[CORE[x], CORE[y]], CORE[x]],
           not[equal[composite[CORE[x], CORE[y]], CORE[intersection[x, y]]]],
           not[equal[Uclosure[x], Uclosure[intersection[x, y]]]]] == True

In[30]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed
```

The three lemmas can be combined to derive:

```
In[31]:= Map[not, SubstTest[and, implies[p1, p2],
           implies[p2, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
             {p1 -> subclass[x, y], p2 -> equal[Uclosure[x], Uclosure[intersection[x, y]]],
                 p3 -> equal[composite[CORE[x], CORE[y]], CORE[intersection[x, y]]],
                 p4 -> equal[composite[CORE[x], CORE[y]], CORE[x]]}]]

Out[31]= or[equal[composite[CORE[x], CORE[y]], CORE[x]], not[subclass[x, y]]] == True

In[32]:= or[equal[composite[CORE[x_], CORE[y_]], CORE[x_]], not[subclass[x_, y_]]] := True
```

the composite in the reverse order

In this section, the case **subclass[x,y]** is considered again, but this time we study the composite of **CORE[x]** and **CORE[y]** in the reverse order. For this case, we redo an argument about **core[x,y]**. which was based on the inclusion **subclass[w, P[U[w]]**.

```
In[33]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → intersection[x, P[z]], v → P[core[x, z]], w → composite[inverse[E], id[y]]}]

Out[33]= subclass[U[intersection[x, y, P[z]]],
           U[intersection[y, P[U[intersection[x, P[z]]]]]]] == True

In[34]:= (% /. {x → x_, y → y_, z -> z_}) /. Equal → SetDelayed
```

Reformulation:

```
In[35]:= subclass[core[intersection[x, y], z], core[y, core[x, z]]]

Out[35]= True
```

An analog of an earlier lemma is redone:

```
In[36]:= SubstTest[implies, subclass[x, w],
          subclass[core[x, z], core[w, z]], w -> intersection[x, y]]

Out[36]= or[not[subclass[x, y]],
           subclass[U[intersection[x, P[z]]], U[intersection[x, y, P[z]]]]] == True

In[37]:= (% /. {x → x_, y → y_, z -> z_}) /. Equal → SetDelayed
```

This yields an inclusion:

```
In[38]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], not[implies[p1, p3]],
             {p1 -> subclass[x, y], p2 -> subclass[core[x, z], core[intersection[x, y], z]],
                 p3 -> subclass[core[x, z], core[y, core[x, z]]]}]]

Out[38]= or[not[subclass[x, y]], subclass[U[intersection[x, P[z]]],
            U[intersection[y, P[U[intersection[x, P[z]]]]]]]] == True

In[39]:= (% /. {x → x_, y → y_, z -> z_}) /. Equal → SetDelayed
```

The reverse inclusion is a general property of **core**, so an equation can be derived:

```
In[40]:= SubstTest[and, implies[p, subclass[u, v]], implies[p, subclass[v, u]],
             {p -> subclass[x, y], u -> core[x, z], v -> core[y, core[x, z]]}] // Reverse

Out[40]= or[equal[U[intersection[x, P[z]]], U[intersection[y, P[U[intersection[x, P[z]]]]]]],
           not[subclass[x, y]]] == True

In[41]:= (% /. {x → x_, y → y_, z -> z_}) /. Equal → SetDelayed
```

In order to derive the functional version of this result, it is convenient to reformulate it without hypotheses:

```
In[42]:= SubstTest[implies, subclass[w, x],
          equal[core[w, z], core[x, core[w, z]]], w -> intersection[x, y]]

Out[42]= equal[U[intersection[x, P[U[intersection[x, y, P[z]]]]]],
           U[intersection[x, y, P[z]]]] == True

In[43]:= U[intersection[x_, P[U[intersection[x_, y_, P[z_]]]]]] := U[intersection[x, y, P[z]]]
```

The functional version is now immediate:

```
In[44]:= composite[CORE[x], CORE[intersection[x, y]]] // VSNormality

Out[44]= composite[CORE[x], CORE[intersection[x, y]]] == CORE[intersection[x, y]]

In[45]:= composite[CORE[x_], CORE[intersection[x_, y_]]] := CORE[intersection[x, y]]
```

As before, one can rewrite this without intersections. To do so, one needs analogs of some earlier lemmas:

```
In[46]:= SubstTest[implies, equal[u, v], equal[composite[w, u], composite[w, v]],
           {u -> CORE[x], v -> CORE[intersection[x, y]], w -> CORE[y]}]

Out[46]= or[equal[composite[CORE[y], CORE[x]], CORE[intersection[x, y]]],
           not[equal[Uclosure[x], Uclosure[intersection[x, y]]]]] == True

In[47]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[48]:= SubstTest[implies, and[equal[u, v], equal[v, w]], equal[u, w],
           {u -> CORE[x], v -> CORE[intersection[x, y]], w -> composite[CORE[y], CORE[x]]}]

Out[48]= or[equal[composite[CORE[y], CORE[x]], CORE[x]],
           not[equal[composite[CORE[y], CORE[x]], CORE[intersection[x, y]]]],
           not[equal[Uclosure[x], Uclosure[intersection[x, y]]]]] == True

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

These lemmas can be combined to yield the desired result:

```
In[50]:= Map[not, SubstTest[and, implies[p1, p2],
          implies[p2, p3], implies[and[p2, p3], p4], not[implies[p1, p4]],
            {p1 -> subclass[x, y], p2 -> equal[Uclosure[x], Uclosure[intersection[x, y]]],
               p3 -> equal[composite[CORE[y], CORE[x]], CORE[intersection[x, y]]],
               p4 -> equal[composite[CORE[y], CORE[x]], CORE[x]]}]]

Out[50]= or[equal[composite[CORE[y], CORE[x]], CORE[x]], not[subclass[x, y]]] == True

In[51]:= or[equal[composite[CORE[y_], CORE[x_]], CORE[x_]], not[subclass[x_, y_]]] := True
```

In particular, it follows that if **x** is a subclass of **y**, or vice−versa, then **CORE[x]** and **CORE[y]** commute.

a special case

Since **x** is contained in **P[U[x]]**, one finds in particular:

```
In[52]:= SubstTest[composite, CORE[intersection[x, y]], CORE[y], y -> P[U[x]]]

Out[52]= composite[CORE[x], IMAGE[id[U[x]]]] == CORE[x]
```

This result can also be derived another way which does not depend on the other results derived in this notebook:

```
In[53]:= SubstTest[CORE, intersection[x, P[y]], y -> U[x]] // Reverse

Out[53]= composite[CORE[x], IMAGE[id[U[x]]]] == CORE[x]

In[54]:= composite[CORE[x_], IMAGE[id[U[x_]]]] := CORE[x]
```

No additional work is needed for the composite in the reverse order:

```
In[55]:= composite[IMAGE[id[U[x]]], CORE[x]]

Out[55]= CORE[x]
```

For completeness, we note the following fact which is closely related to the above results:

```
In[56]:= SubstTest[implies, subclass[x, y],
          subclass[CORE[y], composite[S, CORE[x]]], y -> P[U[x]]]

Out[56]= subclass[IMAGE[id[U[x]]], composite[S, CORE[x]]] == True

In[57]:= subclass[IMAGE[id[U[x_]]], composite[S, CORE[x_]]] := True
```

CORE[union[x,y]]

A formula for **CORE[union[x,y]]** is readily obtained by applying **RelnNormality**. To clean it up, a lemma is needed:

```
In[58]:= composite[CUP, intersection[composite[inverse[FIRST], CORE[x]],
            composite[inverse[SECOND], CORE[y]]]] // VSNormality // Reverse

Out[58]= intersection[composite[S, CORE[x]],
           composite[S, CORE[y]], composite[inverse[S], CUP, intersection[
             composite[inverse[FIRST], CORE[x]], composite[inverse[SECOND], CORE[y]]]]] ==
          composite[CUP, intersection[composite[inverse[FIRST], CORE[x]],
            composite[inverse[SECOND], CORE[y]]]]

In[59]:= intersection[composite[S, CORE[x_]],
           composite[S, CORE[y_]], composite[inverse[S], CUP, intersection[
             composite[inverse[FIRST], CORE[x_]], composite[inverse[SECOND], CORE[y_]]]]] :=
          composite[CUP, intersection[composite[inverse[FIRST], CORE[x]],
            composite[inverse[SECOND], CORE[y]]]]
```

The main theorem is:

```
In[60]:= CORE[union[x, y]] // RelnNormality // Reverse

Out[60]= composite[CUP, intersection[composite[inverse[FIRST], CORE[x]],
            composite[inverse[SECOND], CORE[y]]]] == CORE[union[x, y]]
```

The following orientation of this equation as a rewrite rule is tentatively adopted:

```
In[61]:= composite[CUP, intersection[composite[inverse[FIRST], CORE[x_]],
            composite[inverse[SECOND], CORE[y_]]]] := CORE[union[x, y]]
```

Restatement:

```
In[62]:= composite[CUP, cross[CORE[x], CORE[y]], DUP] == CORE[union[x, y]]

Out[62]= True
```

The following related result is also of interest:

```
In[63]:= composite[S, CORE[union[x, y]]] // VSNormality // Reverse

Out[63]= intersection[composite[S, CORE[x]], composite[S, CORE[y]]] ==
          composite[S, CORE[union[x, y]]]
```

The following orientation of this equation as a rewrite rule is tentatively adopted:

```
In[64]:= intersection[composite[S, CORE[x_]], composite[S, CORE[y_]]] :=
          composite[S, CORE[union[x, y]]]
```

Corollary:

```
In[65]:= SubstTest[composite, CUP, cross[CORE[x], CORE[y]], DUP, y -> V]

Out[65]= composite[CUP, id[CORE[x]], inverse[FIRST]] == Id

In[66]:= composite[CUP, id[CORE[x_]], inverse[FIRST]] := Id
```

a special case

Note that:

```
In[67]:= CORE[P[x]]

Out[67]= IMAGE[id[x]]
```

If follows that any result about **CORE[x]** implies a correponding result about the special case **IMAGE[id[x]]**. One can also derive this result independently:

```
In[68]:= symdif[IMAGE[id[union[x, y]]],
           composite[CUP, cross[IMAGE[id[x]], IMAGE[id[y]]], DUP]] // VSNormality

Out[68]= union[intersection[complement[IMAGE[id[union[x, y]]]],
            composite[CUP, intersection[composite[inverse[FIRST], IMAGE[id[x]]],
              composite[inverse[SECOND], IMAGE[id[y]]]]]], intersection[
            composite[complement[CUP], intersection[composite[inverse[FIRST], IMAGE[id[x]]],
              composite[inverse[SECOND], IMAGE[id[y]]]]], IMAGE[id[union[x, y]]]]] == 0

In[69]:= (% /. {x -> x_, y -> y_}) /. Equal -> SetDelayed

In[70]:= SubstTest[equal, 0, symdif[u, v], {u -> IMAGE[id[union[x, y]]],
            v -> composite[CUP, cross[IMAGE[id[x]], IMAGE[id[y]]], DUP]}] // Reverse

Out[70]= equal[composite[CUP, intersection[composite[inverse[FIRST], IMAGE[id[x]]],
             composite[inverse[SECOND], IMAGE[id[y]]]]], IMAGE[id[union[x, y]]]] == True

In[71]:= composite[CUP, intersection[composite[inverse[FIRST], IMAGE[id[x_]]],
            composite[inverse[SECOND], IMAGE[id[y_]]]]] := IMAGE[id[union[x, y]]]
```

Comparing this with the general theorem, one finds a curious result:

```
In[72]:= SubstTest[composite, CUP, cross[CORE[u], CORE[v]], DUP, {u -> P[x], v -> P[y]}] // Reverse

Out[72]= CORE[union[P[x], P[y]]] == IMAGE[id[union[x, y]]]

In[73]:= CORE[union[P[x_], P[y_]]] := IMAGE[id[union[x, y]]]
```

Corollary 1.

```
In[74]:= SubstTest[intersection, composite[S, CORE[u]],
          composite[S, CORE[v]], {u -> P[x], v -> P[y]}]

Out[74]= intersection[composite[S, IMAGE[id[x]]], composite[S, IMAGE[id[y]]]] ==
          composite[S, IMAGE[id[union[x, y]]]]

In[75]:= intersection[composite[S, IMAGE[id[x_]]], composite[S, IMAGE[id[y_]]]] :=
          composite[S, IMAGE[id[union[x, y]]]]
```

Corollary 2.

```
In[76]:= SubstTest[range, CORE[z], z -> union[P[x], P[y]]] // Reverse

Out[76]= Uclosure[union[P[x], P[y]]] == P[union[x, y]]
```

This second corollary can also be derived another way:

```
In[77]:= SubstTest[Uclosure, image[inverse[S], z], z -> union[P[x], P[y]]]

Out[77]= Uclosure[union[P[x], P[y]]] == P[union[x, y]]

In[78]:= Uclosure[union[P[x_], P[y_]]] := P[union[x, y]]
```

The following older formula for the power class of a union seems to be closely related to the new formula:

```
In[79]:= image[CUP, cart[P[x], P[y]]]

Out[79]= P[union[x, y]]
```