binop[x] ⊂ COMPOSE

Johan G. F. Belinfante
2011 October 7

In[1]:= **SetDirectory["l:"]; << goedel.11oct04b**

```
:Package Title: goedel.11oct04b                2011 October 4 at 3:20 p.m.

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

It is now:  2011 Oct 7 at 16:6

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 Oct 7 at 16:18
```

---

summary

A subset of **COMPOSE** is a binary operation if and only it if it the restriction of **COMPOSE** to the cartesian square of a set which is closed under composition. All such binary operations are semigroups. An example is the addition of non-negative integers.

---

derivation

Theorem. The restriction of **COMPOSE** to **x × x** is a binary operation if and only if **x ∈ binclosed[COMPOSE]**.

In[2]:=
```
SubstTest[and, FUNCTION[t], equal[domain[t], cartsq[fix[domain[t]]]],
  subclass[range[t], fix[domain[t]]], member[t, V],
  t -> composite[COMPOSE, id[cart[x, x]]]]
```

Out[2]=
```
member[composite[COMPOSE, id[cart[x, x]]], BINOPS] ==
  and[member[x, V], subclass[image[COMPOSE, cart[x, x]], x]]
```

In[3]:=
```
member[composite[COMPOSE, id[cart[x_, x_]]], BINOPS] :=
  and[member[x, V], subclass[image[COMPOSE, cart[x, x]], x]]
```

Lemma.

```
In[4]:=  Map[empty, dif[id[binclosed[COMPOSE]], image[inverse[CART], image[
              inverse[IMAGE[composite[id[COMPOSE], inverse[FIRST]]]], BINOPS]]] // Normality]

Out[4]=  subclass[binclosed[COMPOSE], fix[image[inverse[CART],
             image[inverse[IMAGE[composite[id[COMPOSE], inverse[FIRST]]]], BINOPS]]]] == True

In[5]:=  % /. Equal → SetDelayed
```

Theorem. A variable-free restatement.

```
In[6]:=  SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → composite[IMAGE[composite[id[COMPOSE], inverse[FIRST]]], CART],
            u → id[binclosed[COMPOSE]], v -> image[inverse[CART],
              image[inverse[IMAGE[composite[id[COMPOSE], inverse[FIRST]]]], BINOPS]]}] // Reverse

Out[6]=  subclass[image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
            image[CART, id[binclosed[COMPOSE]]]], BINOPS] == True

In[7]:=  subclass[image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
            image[CART, id[binclosed[COMPOSE]]]], BINOPS] := True
```

Observation. The following inclusion follows from this.

```
In[8]:=  subclass[image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
           image[CART, id[binclosed[COMPOSE]]]], intersection[BINOPS, P[COMPOSE]]]

Out[8]=  True
```

The reverse inclusion will be derived in the next section and is combined with the above inclusion to obtain a variable-free equation.

---

the reverse inclusion

The **binop** wrapper will be used to derive the reverse inclusion.

Lemma.

```
In[9]:=  SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
           {t → composite[IMAGE[composite[id[COMPOSE], inverse[FIRST]]]],
            u → set[domain[binop[x]]], v → image[CART, Id]}] // Reverse

Out[9]=  member[composite[COMPOSE, id[domain[binop[x]]]],
           image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]], image[CART, Id]]] == True

In[10]:=  (% /. x → x_) /. Equal → SetDelayed
```

Theorem. If **binop[x]** ⊂ **COMPOSE**, then **binop[x]** is the restriction of **COMPOSE** to **domain[binop[x]]**.

```
In[11]:= Map[implies[subclass[binop[x], COMPOSE], #] &, SubstTest[equal, u,
            composite[funpart[v], id[domain[u]]], {u → binop[x], v → COMPOSE}]] // Reverse

Out[11]= or[equal[binop[x], composite[COMPOSE, id[domain[binop[x]]]]],
          not[subclass[binop[x], COMPOSE]]] == True

In[12]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem.

```
In[13]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            not[implies[p1, p3]], {p1 -> subclass[binop[x], COMPOSE],
             p2 -> equal[binop[x], composite[COMPOSE, id[domain[binop[x]]]]],
             p3 -> member[binop[x], image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
                image[CART, Id]]]}]] // Reverse

Out[13]= or[member[binop[x],
            image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]], image[CART, Id]]],
          not[subclass[binop[x], COMPOSE]]] == True

In[14]:= (% /. x → x_) /. Equal → SetDelayed
```

Corollary.

```
In[15]:= Map[equal[V, domain[#]] &,
          SubstTest[reify, x, case[or[member[binop[x], u], not[subclass[binop[x], v]]]],
           {u -> image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]], image[CART, Id]],
            v → COMPOSE}]]

Out[15]= subclass[intersection[BINOPS, P[COMPOSE]],
          image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]], image[CART, Id]]] == True

In[16]:= subclass[intersection[BINOPS, P[COMPOSE]],
          image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]], image[CART, Id]]] := True
```

Theorem.

```
In[17]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            not[implies[p1, p3]], {p1 -> subclass[binop[x], COMPOSE],
             p2 -> equal[binop[x], composite[COMPOSE, id[domain[binop[x]]]]],
             p3 -> member[composite[COMPOSE, id[domain[binop[x]]]], BINOPS]}]] // Reverse

Out[17]= or[member[composite[COMPOSE, id[domain[binop[x]]]], BINOPS],
          not[subclass[binop[x], COMPOSE]]] == True

In[18]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[19]:= SubstTest[implies, member[t, BINOPS], subclass[range[t], fix[domain[t]]],
          t -> composite[COMPOSE, id[domain[binop[x]]]]] // Reverse

Out[19]= or[not[member[composite[COMPOSE, id[domain[binop[x]]]], BINOPS]],
          subclass[image[COMPOSE, domain[binop[x]]], fix[domain[binop[x]]]]] == True
```

```
In[20]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. If **binop[x] ⊂ COMPOSE**, then **fix[domain[binop[x]]]** is closed under composition.

```
In[21]:= Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3],
            not[implies[p1, p3]], {p1 -> subclass[binop[x], COMPOSE],
             p2 -> member[composite[COMPOSE, id[domain[binop[x]]]], BINOPS], p3 ->
              subclass[image[COMPOSE, domain[binop[x]]], fix[domain[binop[x]]]]}]] // Reverse

Out[21]= or[not[subclass[binop[x], COMPOSE]],
          subclass[image[COMPOSE, domain[binop[x]]], fix[domain[binop[x]]]]] == True

In[22]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma.

```
In[23]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
          {t → composite[IMAGE[composite[id[COMPOSE], inverse[FIRST]]]],
           u → set[domain[binop[x]]], v → image[CART, id[binclosed[COMPOSE]]]}] // Reverse

Out[23]= or[member[composite[COMPOSE, id[domain[binop[x]]]],
            image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
             image[CART, id[binclosed[COMPOSE]]]]],
          not[subclass[image[COMPOSE, domain[binop[x]]], fix[domain[binop[x]]]]]] == True

In[24]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. The promised reverse inclusion.

```
In[25]:= Map[equal[V, domain[reify[x, case[#]]]] &,
          Map[not, SubstTest[and, implies[p1, p2], implies[p2, p3], implies[p1, p4],
            implies[and[p3, p4], p5], not[implies[p1, p5]], {p1 → subclass[binop[x], COMPOSE],
             p2 -> subclass[image[COMPOSE, domain[binop[x]]], fix[domain[binop[x]]]],
             p3 -> member[composite[COMPOSE, id[domain[binop[x]]]], image[IMAGE[composite[
                  id[COMPOSE], inverse[FIRST]]], image[CART, id[binclosed[COMPOSE]]]]],
             p4 → equal[binop[x], composite[COMPOSE, id[domain[binop[x]]]]],
             p5 -> member[binop[x], image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
                image[CART, id[binclosed[COMPOSE]]]]]}]]] // Reverse

Out[25]= subclass[intersection[BINOPS, P[COMPOSE]],
          image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
           image[CART, id[binclosed[COMPOSE]]]]] == True

In[26]:= % /. Equal → SetDelayed
```

Main Theorem. A variable-free equation.

```
In[27]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
             image[CART, id[binclosed[COMPOSE]]]], v -> intersection[BINOPS, P[COMPOSE]]}]

Out[27]= equal[image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
            image[CART, id[binclosed[COMPOSE]]]], intersection[BINOPS, P[COMPOSE]]] == True
```

```
In[28]:= image[IMAGE[composite[id[COMPOSE], inverse[FIRST]]],
           image[CART, id[binclosed[COMPOSE]]]] := intersection[BINOPS, P[COMPOSE]]
```

semigroup result

Lemma.

```
In[29]:= SubstTest[implies, and[associative[t], subclass[image[t, cart[x, x]], x]],
           associative[composite[t, id[cart[x, x]]]]], t → COMPOSE] // Reverse

Out[29]= or[associative[composite[COMPOSE, id[cart[x, x]]]],
           not[subclass[image[COMPOSE, cart[x, x]], x]]] == True

In[30]:= or[associative[composite[COMPOSE, id[cart[x_, x_]]]],
           not[subclass[image[COMPOSE, cart[x_, x_]], x_]]] := True
```

Lemma.

```
In[31]:= Map[implies[and[member[x, V], subclass[image[COMPOSE, cart[x, x]], x]], #] &, SubstTest[
           and, associative[t], member[t, BINOPS], t -> composite[COMPOSE, id[cart[x, x]]]]]]

Out[31]= or[member[composite[COMPOSE, id[cart[x, x]]], SEMIGPS],
           not[member[x, V]], not[subclass[image[COMPOSE, cart[x, x]], x]]] == True

In[32]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. (Converse result.)

```
In[33]:= SubstTest[implies, member[t, SEMIGPS],
           member[t, BINOPS], t -> composite[COMPOSE, id[cart[x, x]]]] // Reverse

Out[33]= or[and[member[x, V], subclass[image[COMPOSE, cart[x, x]], x]],
           not[member[composite[COMPOSE, id[cart[x, x]]], SEMIGPS]]] == True

In[34]:= (% /. x → x_) /. Equal → SetDelayed
```

Theorem. The restriction of **COMPOSE** to **x** × **x** is a semigroup if and only if **x** ∈ **binclosed[COMPOSE]**.

```
In[35]:= equiv[member[composite[COMPOSE, id[cart[x, x]]], SEMIGPS],
          and[member[x, V], subclass[image[COMPOSE, cart[x, x]], x]]]

Out[35]= True

In[36]:= member[composite[COMPOSE, id[cart[x_, x_]]], SEMIGPS] :=
          and[member[x, V], subclass[image[COMPOSE, cart[x, x]], x]]
```

Lemma.

```
In[37]:= Map[empty, dif[id[binclosed[COMPOSE]], image[inverse[CART], image[
              inverse[IMAGE[composite[id[COMPOSE], inverse[FIRST]]]], SEMIGPS]]] // Normality]

Out[37]= subclass[binclosed[COMPOSE], fix[image[inverse[CART],
             image[inverse[IMAGE[composite[id[COMPOSE], inverse[FIRST]]]], SEMIGPS]]]] == True

In[38]:= % /. Equal → SetDelayed
```

Theorem. Any binary operation which is a subset of **COMPOSE** is a semigroup.

```
In[39]:= Map[or[#, subclass[intersection[BINOPS, P[COMPOSE]], SEMIGPS]] &,
           SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
            {t → composite[IMAGE[composite[id[COMPOSE], inverse[FIRST]]], CART],
             u → id[binclosed[COMPOSE]], v -> image[inverse[CART], image[
                inverse[IMAGE[composite[id[COMPOSE], inverse[FIRST]]]], SEMIGPS]]}]] // Reverse

Out[39]= subclass[intersection[BINOPS, P[COMPOSE]], SEMIGPS] == True

In[40]:= subclass[intersection[BINOPS, P[COMPOSE]], SEMIGPS] := True
```

Corollary. An equation.

```
In[41]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> intersection[SEMIGPS, P[COMPOSE]], v -> intersection[BINOPS, P[COMPOSE]]}]

Out[41]= equal[intersection[BINOPS, P[COMPOSE]], intersection[SEMIGPS, P[COMPOSE]]] == True

In[42]:= intersection[SEMIGPS, P[COMPOSE]] := intersection[BINOPS, P[COMPOSE]]
```

addition of non-negative integers

The non-negative integers form a semigroup under addition.

```
In[43]:= member[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]], SEMIGPS]

Out[43]= True
```

Theorem. This semigroup is commutative.

```
In[44]:= Map[member[#, COMMUTATIVE] &,
           Assoc[INTADD, cross[PLUS, PLUS], inverse[cross[PLUS, PLUS]]]] // Reverse

Out[44]= equal[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]],
           composite[COMPOSE, SWAP, id[cart[range[PLUS], range[PLUS]]]]] == True

In[45]:= composite[COMPOSE, SWAP, id[cart[range[PLUS], range[PLUS]]]] :=
          composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]
```

Lemma.

```
In[46]:= Map[member[id[omega], range[#]] &,
           SubstTest[fix, composite[inverse[FIRST], Di, COMPOSE, SWAP, id[cart[x, x]]],
            x → range[PLUS]]] // Reverse

Out[46]= member[id[omega],
           image[fix[composite[inverse[FIRST], Di, COMPOSE]], range[PLUS]]] == False

In[47]:= % /. Equal → SetDelayed
```

Theorem. The integer zero is a neutral element for addition.

```
In[48]:= Map[member[id[omega], #] &, SubstTest[intersection, fix[domain[w]],
           complement[domain[fix[composite[inverse[SECOND], Di, w]]]],
           complement[range[fix[composite[inverse[FIRST], Di, w]]]],
           w -> composite[COMPOSE, id[cartsq[range[PLUS]]]]]]

Out[48]= member[id[omega], ids[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]]] == True

In[49]:= % /. Equal → SetDelayed
```

Theorem.

```
In[50]:= Map[member[id[omega], #] &,
          SubstTest[ids, binop[t], t -> composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]]]

Out[50]= equal[e[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]], id[omega]] == True

In[51]:= e[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]] := id[omega]
```

Theorem.

```
In[52]:= SubstTest[ids, binop[t],
           t -> composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]] // Reverse

Out[52]= ids[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]] == set[id[omega]]

In[53]:= ids[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]] := set[id[omega]]
```

Lemma. Addition is associative.

```
In[54]:= SubstTest[associative, semigp[t],
           t -> composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]] // Reverse

Out[54]= associative[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]] == True

In[55]:= associative[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]]] := True
```

Theorem. The non-negative integers form a monoid under addition.

```
In[56]:= member[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]], MONOIDS] // AssertTest

Out[56]= member[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]], MONOIDS] == True

In[57]:= member[composite[COMPOSE, id[cart[range[PLUS], range[PLUS]]]], MONOIDS] := True
```