unital subsemigroups

Johan G. F. Belinfante
2011 May 22

```
In[1]:= SetDirectory["l:"]; << goedel.11may21a

:Package Title: goedel.11may21a             2011 May 21 at 12:20 noon

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

It is now:  2011 May 22 at 19:20

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 May 22 at 19:31
```

summary

A subsemigroup **x** of a binary operation **y** is said to be **unital** if **y** has a neutral (or unity) element **e[y]** and this element belongs to **fix[domain[x]]**. In this notebook various theorems are derived about unital subsemigroups of monoids and groups.

ids[x]

From the definition of the class **ids[x]** of neutral (or identity) elements, one obtains the following characterization.

Lemma. Characterization of neutral elements.

```
In[2]:= SubstTest[member, x, intersection[u, v, w],
          {u -> fix[domain[y]], v -> complement[domain[fix[composite[inverse[SECOND], Di, y]]]],
           w -> complement[range[fix[composite[inverse[FIRST], Di, y]]]]}]

Out[2]= and[member[x, fix[domain[y]]], subclass[composite[y, LEFT[x]], Id],
          subclass[composite[y, RIGHT[x]], Id]] == member[x, ids[y]]

In[3]:= and[member[x_, fix[domain[y_]]], subclass[composite[y_, LEFT[x_]], Id],
          subclass[composite[y_, RIGHT[x_]], Id]] := member[x, ids[y]]
```

Corollary.

```
In[4]:= or[member[x, ids[y]], not[member[x, fix[domain[y]]]],
          not[subclass[composite[y, LEFT[x]], Id]],
          not[subclass[composite[y, RIGHT[x]], Id]]] // NotNotTest

Out[4]= or[member[x, ids[y]], not[member[x, fix[domain[y]]]],
          not[subclass[composite[y, LEFT[x]], Id]],
          not[subclass[composite[y, RIGHT[x]], Id]]] == True

In[5]:= or[member[x_, ids[y_]], not[member[x_, fix[domain[y_]]]],
          not[subclass[composite[y_, LEFT[x_]], Id]],
          not[subclass[composite[y_, RIGHT[x_]], Id]]] := True
```

A binary operation has at most one neutral element.

```
In[6]:= ids[binop[x]]

Out[6]= set[e[binop[x]]]
```

If **binop[x]** has a neutral element, then it is **e[binop[x]] ∈ fix[domain[binop[x]]]**, and if not, then **e[binop[x]] = V**.

```
In[19]:= (SubstTest[equal, V, A[t], t → ids[w]] /. w → binop[x]) // Reverse

Out[19]= equal[V, e[binop[x]]] == not[member[e[binop[x]], V]]

In[20]:= equal[V, e[binop[x_]]] := not[member[e[binop[x]], V]]
```

Lemma.

```
In[21]:= SubstTest[subclass, ids[t], fix[domain[t]], t → binop[x]] // Reverse

Out[21]= or[member[e[binop[x]], fix[domain[binop[x]]]], not[member[e[binop[x]], V]]] == True

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

Theorem. A simplification rule.

```
In[23]:= equiv[member[e[binop[x]], fix[domain[binop[x]]]], member[e[binop[x]], V]]

Out[23]= True

In[25]:= member[e[binop[x_]], fix[domain[binop[x_]]]] := member[e[binop[x]], V]
```

monoids

A **monoid** **x** is a semigroup with a neutral element **e[x]**.

Lemma.

```
In[26]:= SubstTest[subclass, ids[t], fix[composite[t, DUP]], t → semigp[x]] // Reverse

Out[26]= or[member[pair[pair[e[semigp[x]], e[semigp[x]]], e[semigp[x]]], semigp[x]],
           not[member[e[semigp[x]], V]]] == True
```

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

Theorem. The neutral element of a monoid is idempotent.

```
In[28]:= SubstTest[implies, equal[x, semigp[t]], or[member[pair[pair[e[x], e[x]], e[x]], x],
           not[member[e[x], V]]], t → x] // Reverse // MapNotNot

Out[28]= or[member[pair[pair[e[x], e[x]], e[x]], x], not[member[x, MONOIDS]]] == True

In[29]:= or[member[pair[pair[e[x_], e[x_]], e[x_]], x_], not[member[x_, MONOIDS]]] := True
```

unital subsemigroups of monoids

Observation.

```
In[30]:= implies[subclass[x, y], subclass[intersection[fix[domain[x]], ids[y]], ids[x]]]

Out[30]= True
```

Lemma. (Specialization to the case that both **x** and **y** are semigroups.)

```
In[31]:= SubstTest[implies, subclass[u, v],
          subclass[intersection[fix[domain[u]], ids[v]], ids[u]],
          {u → semigp[x], v → semigp[y]}] // Reverse

Out[31]= or[equal[e[semigp[x]], e[semigp[y]]],
          not[member[e[semigp[y]], fix[domain[semigp[x]]]]],
          not[subclass[semigp[x], semigp[y]]]] == True

In[32]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. (Eliminate the **semigp** wrappers.)

```
In[33]:= SubstTest[implies, and[equal[x, semigp[u]], equal[y, semigp[v]]], or[equal[e[x], e[y]],
           not[member[e[y], fix[domain[x]]]], not[subclass[x, y]]], {u → x, v → y}] // Reverse

Out[33]= or[equal[e[x], e[y]], not[member[x, SEMIGPS]], not[member[y, SEMIGPS]],
          not[member[e[y], fix[domain[x]]]], not[subclass[x, y]]] == True

In[34]:= or[equal[e[x_], e[y_]], not[member[x_, SEMIGPS]], not[member[y_, SEMIGPS]],
          not[member[e[y_], fix[domain[x_]]]], not[subclass[x_, y_]]] := True
```

A semigroup can only have a unital subsemigroup if it is itself a monoid.

Theorem.

```
In[35]:= Map[not, SubstTest[and, implies[p2, p3],
           not[implies[p1, p3]], {p1 → member[e[x], y], p2 → member[e[x], V],
            p3 -> or[member[x, MONOIDS], not[member[x, SEMIGPS]]]}]] // Reverse

Out[35]= or[member[x, MONOIDS], not[member[x, SEMIGPS]], not[member[e[x], y]]] == True
```

```
In[36]:= or[member[x_, MONOIDS], not[member[x_, SEMIGPS]], not[member[e[x_], y_]]] := True
```

Corollary. Any unital semigroup of a semigroup is a monoid.

```
In[37]:= Map[not, SubstTest[and, implies[and[p1, p2, p3, p4], p5], implies[and[p4, p5], p6],
           implies[and[p1, p6], p7], not[implies[and[p1, p2, p3, p4], p7]],
           {p1 → member[x, SEMIGPS], p2 → subclass[x, y], p3 → member[y, SEMIGPS],
            p4 → member[e[y], fix[domain[x]]], p5 → equal[e[x], e[y]],
            p6 → member[e[x], V], p7 → member[x, MONOIDS]}]] // Reverse

Out[37]= or[member[x, MONOIDS], not[member[x, SEMIGPS]], not[member[y, SEMIGPS]],
          not[member[e[y], fix[domain[x]]]], not[subclass[x, y]]] == True

In[38]:= or[member[x_, MONOIDS], not[member[x_, SEMIGPS]], not[member[y_, SEMIGPS]],
          not[member[e[y_], fix[domain[x_]]]], not[subclass[x_, y_]]] := True
```

Corollary. Any unital semigroup of a monoid is a submonoid.

```
In[39]:= Map[not, SubstTest[and, implies[p2, p3], not[implies[p1, p3]], {p1 → member[y, MONOIDS],
            p2 → member[y, SEMIGPS], p3 -> or[member[x, MONOIDS], not[member[x, SEMIGPS]],
              not[member[e[y], fix[domain[x]]]], not[subclass[x, y]]]}]] // Reverse

Out[39]= or[member[x, MONOIDS], not[member[x, SEMIGPS]], not[member[y, MONOIDS]],
          not[member[e[y], fix[domain[x]]]], not[subclass[x, y]]] == True

In[40]:= or[member[x_, MONOIDS], not[member[e[y_], fix[domain[x_]]]],
          not[member[x_, SEMIGPS]], not[member[y_, MONOIDS]], not[subclass[x_, y_]]] := True
```

unital subsemigroups of a group

A **group** is a monoid that is also a quasigroup. Any theorem about monoids is therefore true in particular for groups.

Theorem. Any unital subsemigroup of a group is a monoid.

```
In[41]:= Map[not, SubstTest[and, implies[p2, p3], not[implies[p1, p3]], {p1 → member[y, GROUPS],
            p2 → member[y, MONOIDS], p3 -> or[member[x, MONOIDS], not[member[x, SEMIGPS]],
              not[member[e[y], fix[domain[x]]]], not[subclass[x, y]]]}]] // Reverse

Out[41]= or[member[x, MONOIDS], not[member[x, SEMIGPS]], not[member[y, GROUPS]],
          not[member[e[y], fix[domain[x]]]], not[subclass[x, y]]] == True

In[42]:= or[member[x_, MONOIDS], not[member[x_, SEMIGPS]], not[member[y_, GROUPS]],
          not[member[e[y_], fix[domain[x_]]]], not[subclass[x_, y_]]] := True
```

In a group the neutral element is the only idempotent element.

Theorem. Any submonoid of a group is unital.

```
In[43]:= Map[not,
           SubstTest[and, implies[p1, p4], implies[and[p2, p4], p5], implies[and[p3, p5], p6],
            not[implies[and[p1, p2, p3], p6]], {p1 → member[x, MONOIDS], p2 → subclass[x, y],
             p3 → member[y, GROUPS], p4 -> member[pair[pair[e[x], e[x]], e[x]], x],
             p5 -> member[pair[pair[e[x], e[x]], e[x]], y], p6 → equal[e[x], e[y]]}]] // Reverse

Out[43]= or[equal[e[x], e[y]], not[member[x, MONOIDS]],
           not[member[y, GROUPS]], not[subclass[x, y]]] == True

In[44]:= or[equal[e[x_], e[y_]], not[member[x_, MONOIDS]],
           not[member[y_, GROUPS]], not[subclass[x_, y_]]] := True
```

Theorem. For a unital subsemigroup of a group, the identity element of the group is the identity element of the semigroup.

```
In[45]:= Map[not, SubstTest[and, implies[p2, p3], not[implies[p1, p3]], {p1 → member[y, GROUPS],
             p2 → member[y, SEMIGPS], p3 -> or[equal[e[x], e[y]], not[member[x, SEMIGPS]],
               not[member[e[y], fix[domain[x]]]], not[subclass[x, y]]]}]] // Reverse

Out[45]= or[equal[e[x], e[y]], not[member[x, SEMIGPS]], not[member[y, GROUPS]],
           not[member[e[y], fix[domain[x]]]], not[subclass[x, y]]] == True

In[46]:= or[equal[e[x_], e[y_]], not[member[e[y_], fix[domain[x_]]]],
           not[member[x_, SEMIGPS]], not[member[y_, GROUPS]], not[subclass[x_, y_]]] := True
```

---

unital binary closed sets

A class **y** that is binary closed under a binary operation **x** is said to be **unital** if **e[x] ∈ y**. In this section it is shown that the restriction of a group to the cartesian square of a unital binary closed class is a submonoid.

Lemma.

```
In[48]:= SubstTest[or, member[t, MONOIDS], not[member[t, SEMIGPS]],
           not[member[x, GROUPS]], not[member[e[x], fix[domain[t]]]],
           not[subclass[t, x]], t → composite[x, id[cart[y, y]]]] // Reverse

Out[48]= or[member[composite[x, id[cart[y, y]]], MONOIDS],
           not[member[x, GROUPS]], not[member[composite[x, id[cart[y, y]]], SEMIGPS]],
           not[member[e[x], y]], not[member[e[x], fix[domain[x]]]]] == True

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

Theorem. If **x** is a group, if **y** is binary closed under **x** and if **e[x] ∈ y**, then the restriction of **x** to **y × y** is a monoid.

```
In[50]:= Map[not,
           SubstTest[and, implies[p1, p5], implies[and[p1, p3], p4], implies[and[p1, p5], p6],
            (* implies[and[p1,p2,p4,p6],p7], *) not[implies[and[p1, p2, p3], p7]],
            {p1 → member[x, GROUPS], p2 -> member[e[x], y], p3 → subclass[image[x, cart[y, y]], y],
             p4 -> member[composite[x, id[cart[y, y]]], SEMIGPS],
             p5 → equal[range[x], fix[domain[x]]], p6 → member[e[x], fix[domain[x]]],
             p7 -> member[composite[x, id[cart[y, y]]], MONOIDS]}]] // Reverse

Out[50]= or[member[composite[x, id[cart[y, y]]], MONOIDS], not[member[x, GROUPS]],
           not[member[e[x], y]], not[subclass[image[x, cart[y, y]], y]]] == True

In[51]:= or[member[composite[x_, id[cart[y_, y_]]], MONOIDS], not[member[e[x_], y_]],
           not[member[x_, GROUPS]], not[subclass[image[x_, cart[y_, y_]], y_]]] := True
```

serendipity

In this final section some results derived earlier about neutral elements are specialized to the case of groups.

Lemma. (Specialization to groups by introducing **gp** wrappers.)

```
In[52]:= SubstTest[or, member[x, ids[t]],
           not[member[x, fix[domain[t]]]], not[subclass[composite[t, LEFT[x]], Id]],
           not[subclass[composite[t, RIGHT[x]], Id]], t → gp[y]] // Reverse

Out[52]= or[equal[x, e[gp[y]]], not[member[x, range[gp[y]]]],
           not[subclass[composite[gp[y], LEFT[x]], Id]],
           not[subclass[composite[gp[y], RIGHT[x]], Id]]] == True

In[53]:= or[equal[x_, e[gp[y_]]], not[member[x_, range[gp[y_]]]],
           not[subclass[composite[gp[y_], LEFT[x_]], Id]],
           not[subclass[composite[gp[y_], RIGHT[x_]], Id]]] := True
```

Theorem.

```
In[54]:= SubstTest[implies, equal[x, gp[t]],
            subclass[composite[x, LEFT[e[x]]], Id], t → x] // Reverse // MapNotNot

Out[54]= or[not[member[x, GROUPS]], subclass[composite[x, LEFT[e[x]]], Id]] == True

In[55]:= or[not[member[x_, GROUPS]], subclass[composite[x_, LEFT[e[x_]]], Id]] := True
```

Dual Theorem.

```
In[56]:= SubstTest[implies, equal[x, gp[t]],
            subclass[composite[x, RIGHT[e[x]]], Id], t → x] // Reverse // MapNotNot

Out[56]= or[not[member[x, GROUPS]], subclass[composite[x, RIGHT[e[x]]], Id]] == True

In[57]:= or[not[member[x_, GROUPS]], subclass[composite[x_, RIGHT[e[x_]]], Id]] := True
```