sub-binops

Johan G. F. Belinfante
2009 February 24

```
In[1]:= SetDirectory["l:"]; << goedel.09feb23a; << tools.m

        :Package Title: goedel.09feb23a       2009 February 23 at 4:15 p.m.

        It is now:  2009 Feb 24 at 13:59

        Loading Simplification Rules

        TOOLS.M                                    Revised 2009 February 18

        weightlimit = 40
```

summary

The class **intersection[BINOPS, P[x]]** of all binary operations contained in a given binary operation **x** is a set. A formula relating this set to the class **binclosed[x]** of sets closed under **x** is derived in this notebook. If **x** is a binary operation, and if **t** is a set closed under **x**, then the restriction of **x** to the cartesian square of **t** is a binary operation. Conversely, all binary operations contained in a given one are of this form. Because each binary operation **y** contained in **x** is determined by the fixed-point class of its domain, one can set up an explicit correspondence between the binary operations contained in **x** and the sets closed under **x**.

derivation

Lemma. If a class **y** is closed under **binop[x]**, then the restriction of **binop[x]** to the cartesian square of **y** is a binary operation. (This is derived here by simply introducing a **binop[x]** wrapper into an available theorem about the restrictions of binary operation.)

```
In[2]:= SubstTest[implies, and[member[t, BINOPS], subclass[image[t, cart[y, y]], y]],
          member[composite[t, id[cart[y, y]]], BINOPS], t → binop[x]] // Reverse

Out[2]= or[member[composite[binop[x], id[cart[y, y]]], BINOPS],
          not[subclass[image[binop[x], cart[y, y]], y]]] == True

In[3]:= or[member[composite[binop[x_], id[cart[y_, y_]]], BINOPS],
          not[subclass[image[binop[x_], cart[y_, y_]], y_]]] := True
```

Lemma. (Eliminate the variable **y** from the preceding lemma.)

```
In[4]:= Map[equal[V, #] &, SubstTest[class, y, implies[member[y, u], member[y, v]],
          {u -> binclosed[binop[x]], v -> fix[image[inverse[CART],
              image[inverse[IMAGE[composite[id[binop[x]], inverse[FIRST]]]], BINOPS]]}]]

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

In[5]:= (% /. x → x_) /. Equal → SetDelayed
```

Lemma. A temporary rewrite rule.

```
In[6]:= ImageComp[composite[IMAGE[composite[id[x], inverse[FIRST]]], CART, DUP],
          inverse[composite[IMAGE[composite[id[x], inverse[FIRST]]], CART, DUP]], y] // Reverse

Out[6]= image[IMAGE[composite[id[x], inverse[FIRST]]],
          image[CART, id[fix[image[inverse[CART], image[
               inverse[IMAGE[composite[id[x], inverse[FIRST]]]], y]]]]] ==
         intersection[y, image[IMAGE[composite[id[x], inverse[FIRST]]], image[CART, Id]]]

In[7]:= image[IMAGE[composite[id[x_], inverse[FIRST]]], image[CART, id[fix[image[inverse[CART],
              image[inverse[IMAGE[composite[id[x_], inverse[FIRST]]]], y_]]]]] :=
         intersection[y, image[IMAGE[composite[id[x], inverse[FIRST]]], image[CART, Id]]]
```

Theorem. An improved formulation of the result obtained by eliminating the variable **y**. This inclusion will be subsumed by an equation for the left hand side to be derived in the sequel.

```
In[8]:= SubstTest[implies, subclass[u, v], subclass[image[t, u], image[t, v]],
          {t -> composite[IMAGE[composite[id[binop[x]], inverse[FIRST]]], CART, DUP],
           u -> binclosed[binop[x]], v -> fix[image[inverse[CART], image[
               inverse[IMAGE[composite[id[binop[x]], inverse[FIRST]]]], BINOPS]]}] // Reverse

Out[8]= subclass[image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
           image[CART, id[binclosed[binop[x]]]]], BINOPS] == True

In[9]:= (% /. x → x_) /. Equal → SetDelayed
```

To obtain an equation, one needs an inclusion in the reverse direction. Such a result can be obtained by restricting one's attention to restrictions of a given binary operation **binop[x]**.

Lemma. Every subclass of a binary operation is a restriction.

```
In[10]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
           equal[u, composite[v, id[domain[u]]]], {u → w, v → binop[x]}] // Reverse

Out[10]= or[equal[w, composite[binop[x], id[domain[w]]]], not[subclass[w, binop[x]]]] == True

In[11]:= (% /. {w → w_, x → x_}) /. Equal → SetDelayed
```

Theorem. If **y** is a binary operation contained in **binop[x]**, then **y** is the restriction of **binop[x]** to the cartesian square of **fix[domain[y]]**.

```
In[12]:= Map[not, SubstTest[and, implies[p2, p3], implies[p1, p4],
           implies[and[p3, p4], p5], not[implies[and[p1, p2], p5]], {p1 → member[y, BINOPS],
            p2 → subclass[y, binop[x]], p3 → equal[y, composite[binop[x], id[domain[y]]]],
            p4 → equal[cart[fix[domain[y]], fix[domain[y]]], domain[y]], p5 → equal[y,
              composite[binop[x], id[cart[fix[domain[y]], fix[domain[y]]]]]]}]] // Reverse

Out[12]= or[equal[y, composite[binop[x], id[cart[fix[domain[y]], fix[domain[y]]]]]],
          not[member[y, BINOPS]], not[subclass[y, binop[x]]]] == True

In[13]:= or[equal[y_, composite[binop[x_], id[cart[fix[domain[y_]], fix[domain[y_]]]]]],
          not[member[y_, BINOPS]], not[subclass[y_, binop[x_]]]] := True
```

Lemma. (An explicit formula for the range of a restriction of **binop[x]**.)

```
In[14]:= SubstTest[implies, equal[y, t], equal[range[y], range[t]],
          t -> composite[binop[x], id[cart[fix[domain[y]], fix[domain[y]]]]]] // Reverse

Out[14]= or[equal[image[binop[x], cart[fix[domain[y]], fix[domain[y]]]], range[y]],
          not[equal[y, composite[binop[x], id[cart[fix[domain[y]], fix[domain[y]]]]]]] == True

In[15]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem. If a binary operation **y** is contained in **binop[x]**, then **fix[domain[y]]** is closed under **binop[x]**.

```
In[16]:= Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p3, p4],
           implies[p1, p5], implies[and[p4, p5], p6], not[implies[and[p1, p2], p6]],
           {p1 → member[y, BINOPS], p2 → subclass[y, binop[x]],
            p3 → equal[y, composite[binop[x], id[cart[fix[domain[y]], fix[domain[y]]]]]],
            p4 → equal[range[y], image[binop[x], cartsq[fix[domain[y]]]]],
            p5 → subclass[range[y], fix[domain[y]]], p6 -> subclass[image[binop[x],
               cart[fix[domain[y]], fix[domain[y]]]], fix[domain[y]]]}] // Reverse

Out[16]= or[not[member[y, BINOPS]], not[subclass[y, binop[x]]], subclass[
           image[binop[x], cart[fix[domain[y]], fix[domain[y]]]], fix[domain[y]]]] == True

In[17]:= or[not[member[y_, BINOPS]], not[subclass[y_, binop[x_]]], subclass[
           image[binop[x_], cart[fix[domain[y_]], fix[domain[y_]]]], fix[domain[y_]]]] := True
```

Technical Lemma.

```
In[18]:= SubstTest[implies, and[member[u, domain[funpart[t]]],
           equal[y, APPLY[funpart[t], u]], member[u, v]], member[y, image[funpart[t], v]],
          {t -> composite[IMAGE[composite[id[binop[x]], inverse[FIRST]]], CART, DUP],
           u → fix[domain[y]], v → binclosed[binop[x]]}] // Reverse

Out[18]= or[member[y, image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
            image[CART, id[binclosed[binop[x]]]]]],
          not[equal[y, composite[binop[x], id[cart[fix[domain[y]], fix[domain[y]]]]]]],
          not[member[fix[domain[y]], V]], not[subclass[
            image[binop[x], cart[fix[domain[y]], fix[domain[y]]]], fix[domain[y]]]] == True

In[19]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Theorem.

```
In[20]:= Map[not,
           SubstTest[and, implies[and[p1, p2], p3], implies[p1, p4], implies[and[p1, p2], p5],
            not[implies[and[p1, p2], p6]], {p1 → member[y, BINOPS], p2 → subclass[y, binop[x]],
             p3 → equal[y, composite[binop[x], id[cart[fix[domain[y]], fix[domain[y]]]]]],
             p4 -> member[fix[domain[y]], V], p5 ->
              subclass[image[binop[x], cart[fix[domain[y]], fix[domain[y]]]], fix[domain[y]]],
             p6 -> member[y, image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
                image[CART, id[binclosed[binop[x]]]]]]}]] // Reverse

Out[20]= or[member[y, image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
            image[CART, id[binclosed[binop[x]]]]]],
          not[member[y, BINOPS]], not[subclass[y, binop[x]]]] == True

In[21]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

Lemma. Eliminate the variable **y**.

```
In[22]:= Map[equal[V, #] &, dif[intersection[BINOPS, P[binop[x]]],
            image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
             image[CART, id[binclosed[binop[x]]]]]] // complement // Normality]

Out[22]= subclass[intersection[BINOPS, P[binop[x]]],
           image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
            image[CART, id[binclosed[binop[x]]]]]] == True

In[23]:= (% /. x → x_) /. Equal → SetDelayed
```

The two inclusions can be combined into an equation and made into a rewrite rule.

Theorem. The class of binary operations contained in a given binary operation **binop[x]** is the set of all restrictions of binop[x] to cartesian squares of sets that are closed under **binop[x]**.

```
In[24]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> intersection[BINOPS, P[binop[x]]], v -> image[IMAGE[
              composite[id[binop[x]], inverse[FIRST]]], image[CART, id[binclosed[binop[x]]]]]}]

Out[24]= equal[image[IMAGE[composite[id[binop[x]], inverse[FIRST]]],
            image[CART, id[binclosed[binop[x]]]], intersection[BINOPS, P[binop[x]]]] == True

In[25]:= image[IMAGE[composite[id[binop[x_]], inverse[FIRST]]],
           image[CART, id[binclosed[binop[x_]]]]] := intersection[BINOPS, P[binop[x]]]
```

Corollary. (Restatement without the **binop** wrapper.)

```
In[26]:= SubstTest[implies, equal[x, binop[t]],
           equal[intersection[BINOPS, P[x]], image[IMAGE[composite[id[x], inverse[FIRST]]],
             image[CART, id[binclosed[x]]]]], t → x] // Reverse

Out[26]= or[
           equal[image[IMAGE[composite[id[x], inverse[FIRST]]], image[CART, id[binclosed[x]]]],
            intersection[BINOPS, P[x]]], not[member[x, BINOPS]]] == True
```

```
In[27]:= or[equal[
           image[IMAGE[composite[id[x_], inverse[FIRST]]], image[CART, id[binclosed[x_]]]],
           intersection[BINOPS, P[x_]]], not[member[x_, BINOPS]]] := True
```