sethood of intervals

Johan G. F. Belinfante
2004 November 17

```
In[1]:= SetDirectory["i:"]; << goedel63.16a; << tools.m

        :Package Title: goedel63.16a        2004 November 16 at 11:55 a.m.

        It is now:  2004 Nov 17 at 9:30

        Loading Simplification Rules

        TOOLS.M                       Revised 2004 November 17

        weightlimit = 40
```

summary

The class of sets **t** satisfying **subclass[x, t]** and **subclass[t, y]** is the **interval** from **x** to **y**. A necessary and sufficient condition for an interval to be a set is derived in this notebook. As an application of this, a rewrite rule is derived for **domain[-VERTSECT[inverse[RESTRICT]]]]**. The result obtained is intuitively clear. Given any relation **x**, there are many relations of which **x** is a restriction, because one can form the union with any relation whose domain is disjoint from that of **x**. The class of relations which can be adjoined is clearly a proper class, and so the class of relations having **x** as a restriction is a proper class. If **x** is not a relation, then **x** cannot be the restriction of anything, and in this case the class of sets whose restriction is **x** is the empty set. Thus the vertical section of **inverse[RESTRICT]** at **x** is a set if and only if **x** is not a relation.

sethood of intervals

The following trick yields a key special case of the rewrite rule to be derived.

```
In[2]:= SubstTest[member, image[ADJOIN[x], z], V, z → P[y]]

Out[2]= member[intersection[image[S, singleton[x]], P[union[x, y]]], V] ==
         or[member[y, V], not[member[x, V]]]
```

```
In[3]:= member[intersection[image[S, singleton[x_]], P[union[x_, y_]]], V] :=
         or[member[y, V], not[member[x, V]]]
```

Corollary:

```
In[4]:= Map[implies[member[y, z], #] &,
         SubstTest[member, intersection[image[S, singleton[y]], P[union[y, z]]],
           V, z → dif[x, y]] // Reverse]

Out[4]= or[member[intersection[x, complement[y]], V], not[member[y, z]]] ==
         or[member[x, V], not[member[y, z]]]

In[5]:= or[member[intersection[x_, complement[y_]], V], not[member[y_, z_]]] :=
         or[member[x, V], not[member[y, z]]]
```

Lemma.

```
In[6]:= SubstTest[implies, disjoint[u, v], member[intersection[u, v], V],
          {u -> image[S, singleton[x]], v -> P[y]}] // MapNotNot

Out[6]= or[member[intersection[image[S, singleton[x]], P[y]], V], subclass[x, y]] ==
         True

In[7]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

This is a bit tricky, but it works:

```
In[8]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
           implies[and[p3, p4], p5], implies[not[p1], p5],
           not[implies[and[p2, p4], p5]], {p1 → subclass[x, y], p2 → equal[z, dif[y, x]],
            p3 -> equal[y, union[x, z]], p4 → member[dif[y, x], V], p5 ->
             member[intersection[image[S, singleton[x]], P[y]], V]}]] /. z → dif[y, x]

Out[8]= or[member[intersection[image[S, singleton[x]], P[y]], V],
          not[member[intersection[y, complement[x]], V]]] == True

In[9]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

This is similar:

```
In[10]:= Map[not, SubstTest[and, implies[and[p1, p2], p3],
            implies[and[p0, p3, p5], p4], not[implies[and[p0, p1, p2, p5], p4]],
            {p0 → member[x, V], p1 → subclass[x, y], p2 → equal[z, dif[y, x]],
             p3 -> equal[y, union[x, z]], p4 → member[dif[y, x], V], p5 ->
              member[intersection[image[S, singleton[x]], P[y]], V]}]] /. z → dif[y, x]

Out[10]= or[member[y, V], not[member[x, V]],
           not[member[intersection[image[S, singleton[x]], P[y]], V]],
           not[subclass[x, y]]] == True
```

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

The upshot of all this trickery is the following rewrite rule.

```
In[12]:= equiv[member[intersection[image[S, singleton[x]], P[y]], V],
            or[member[y, V], not[member[x, V]], not[subclass[x, y]]]] // not // not

Out[12]= True

In[13]:= member[intersection[image[S, singleton[x_]], P[y_]], V] :=
          or[member[y, V], not[member[x, V]], not[subclass[x, y]]]
```

---

another sethood result

This following sethood rule is also needed for the application in the next section.

```
In[14]:= member[intersection[complement[image[V, y]], x], V] // AssertTest

Out[14]= member[intersection[x, complement[image[V, y]]], V] ==
          or[member[x, V], not[equal[0, y]]]

In[15]:= member[intersection[x_, complement[image[V, y_]]], V] :=
          or[member[x, V], not[equal[0, y]]]
```

---

image[inverse[RESTRICT],singleton[x]]

The starting point for the application to **inverse[RESTRICT]** is a normalization rule which relates its vertical sections to certain intervals.

```
In[16]:= image[inverse[RESTRICT], singleton[x]] // Renormality // Reverse

Out[16]= intersection[complement[image[V, intersection[x, complement[cart[V, V]]]]],
           image[S, singleton[x]],
           P[union[complement[cart[domain[x], V]], composite[Id, x]]]] ==
         image[inverse[image[inverse[COMPOSE], singleton[x]]], P[Id]]

In[17]:= intersection[complement[image[V, intersection[x, complement[cart[V, V]]]]],
           image[S, singleton[x_]],
           P[union[complement[cart[domain[x_], V]], composite[Id, x_]]]] :=
         image[inverse[image[inverse[COMPOSE], singleton[x]]], P[Id]]
```

The sethood results derived in the preceding two sections yields something akin to what is needed:

```
In[18]:= Map[member[#, V] &,
           intersection[complement[image[V, intersection[x, complement[cart[V, V]]]]],
             image[S, singleton[x]],
             P[union[x, complement[cart[domain[x], V]]]]] // Normality] // Reverse

Out[18]= or[member[image[inverse[image[inverse[COMPOSE], singleton[x]]], P[Id]], V],
           not[member[x, V]]] == or[not[member[x, V]], not[subclass[x, cart[V, V]]]]

In[19]:= or[member[image[inverse[image[inverse[COMPOSE], singleton[x_]]], P[Id]], V],
           not[member[x_, V]]] := or[not[member[x, V]], not[subclass[x, cart[V, V]]]]
```

What is actually needed is only slightly different:

```
In[20]:= member[x, domain[VERTSECT[inverse[RESTRICT]]]]

Out[20]= and[member[x, V],
          member[image[inverse[image[inverse[COMPOSE], singleton[x]]], P[Id]], V]]
```

The following derivation yields a more useful rewrite rule:

```
In[21]:= SubstTest[and, p, implies[p, q], {p → member[x, V], q -> member[image[
               inverse[image[inverse[COMPOSE], singleton[x]]], P[Id]], V]}] // Reverse

Out[21]= and[member[x, V],
           member[image[inverse[image[inverse[COMPOSE], singleton[x]]], P[Id]], V]] ==
          and[member[x, V], not[subclass[x, cart[V, V]]]]

In[22]:= and[member[x_, V],
           member[image[inverse[image[inverse[COMPOSE], singleton[x_]]], P[Id]], V]] :=
          and[member[x, V], not[subclass[x, cart[V, V]]]]
```

The main result now follows immediately:

```
In[23]:= SubstTest[class, x, member[x, y],
           y -> domain[VERTSECT[inverse[RESTRICT]]]] // Reverse

Out[23]= domain[VERTSECT[inverse[RESTRICT]]] == complement[P[cart[V, V]]]

In[24]:= domain[VERTSECT[inverse[RESTRICT]]] := complement[P[cart[V, V]]]
```