the class RS[x] of small restrictions, part 3

Johan G. F. Belinfante
revised 2004 August 10

```
In[1]:= SetDirectory["i:"]; << goedel60.10a; << tools.m;

:Package Title: goedel60.10a          2004 August 10 at 10:50 p.m.

It is now:  2004 Aug 10 at 23:26

Loading Simplification Rules

TOOLS.M                      Revised 2004 August 08

weightlimit = 40
```

summary

In this third notebook dealing with the theory of **RS[x]**, a study is made of the functon **VERTSECT[RESTRICT] = lambda[x, RS[x]]**. This function allows one to formulate variable-free equations that capture the facts about **RS[x]** for the case that **x** is a set. An additional bonus of this endeavor is the discovery of a cornucopia of new facts about the relation **RESTRICT** which had up to now only been studied cursorily.

the connection between RESTRICT and RS[x]

The relation **RESTRICT** is the class of all ordered pairs **pair[x,y]** such that **y** is a restriction of **x**.

```
In[2]:= class[pair[x, y], exists[z, and[subclass[z, Id], equal[y, composite[x, z]]]]]

Out[2]= RESTRICT
```

The relation **RESTRICT** was actually defined by the following membership rule, which will shortly be replaced.

```
In[3]:= member[pair[x, y], RESTRICT]

Out[3]= and[member[x, V], member[y, image[COMPOSE, cart[singleton[x], P[Id]]]]]
```

The key result needed to connect this relation **RESTRICT** with **RS[x]** is this formula:

```
In[4]:= ImageComp[COMPOSE, LEFT[x], P[Id]] // Reverse

Out[4]= image[COMPOSE, cart[singleton[x], P[Id]]] ==
         intersection[image[V, singleton[x]], RS[x]]

In[5]:= image[COMPOSE, cart[singleton[x_], P[Id]]] :=
         intersection[image[V, singleton[x]], RS[x]]
```

new membership rule for RESTRICT

The key result found in the preceding section causes the membership rule that defined the relation **RESTRICT** to be transformed:

```
In[6]:= member[x, RESTRICT]

Out[6]= and[equal[composite[first[x], id[domain[second[x]]]], second[x]],
         member[first[x], V]]
```

For ordered pairs, this formula becomes rather ugly, so we temporarily remove the old rule, derive a separate rule for membership of pairs, and then restore the above rule.

```
In[7]:= member[x_, RESTRICT] =.
```

Lemma.

```
In[8]:= SubstTest[member, pair[x, y], composite[Id, z], z → RESTRICT] // Reverse

Out[8]= and[member[x, V], member[y, V], member[pair[x, y], RESTRICT]] ==
         member[pair[x, y], RESTRICT]

In[9]:= and[member[x_, V], member[y_, V], member[pair[x_, y_], RESTRICT]] :=
         member[pair[x, y], RESTRICT]
```

This is the new membership rule for pairs:

```
In[10]:= SubstTest[member, y, image[z, singleton[x]], z → RESTRICT] // Reverse

Out[10]= member[pair[x, y], RESTRICT] ==
          and[equal[y, composite[x, id[domain[y]]]], member[x, V], member[y, V]]

In[11]:= member[pair[x_, y_], RESTRICT] :=
          and[equal[y, composite[x, id[domain[y]]]], member[x, V], member[y, V]]
```

The following tricky maneuver restores the replacement for the general membership rule:

```
In[12]:= (member[x, composite[rotate[rotate[w]], id[cart[V, P[Id]]],
              inverse[FIRST]]] // AssertTest) /. w → rotate[COMPOSE]

Out[12]= member[x, RESTRICT] ==
          and[equal[composite[first[x], id[domain[second[x]]]], second[x]],
           member[first[x], V]]

In[13]:= member[x_, RESTRICT] :=
          and[equal[composite[first[x], id[domain[second[x]]]], second[x]],
           member[first[x], V]]
```

---

normalization rule

Subsequent derivations are simplified if one normalizes **RESTRICT**.

```
In[14]:= RESTRICT // VSNormality // Reverse

Out[14]= composite[intersection[inverse[S],
            UB[union[E, composite[inverse[IMAGE[FIRST]], complement[E], FIRST]]]],
           IMAGE[id[cart[V, V]]]] == RESTRICT

In[15]:= composite[intersection[inverse[S],
            UB[union[E, composite[inverse[IMAGE[FIRST]], complement[E], FIRST]]]],
           IMAGE[id[cart[V, V]]]] := RESTRICT
```

---

a partial order related to RESTRICT

The relation **RESTRICT** is idempotent, antisymmetric, and transitive. It fails to be a partial order because it lacks the reflexive property.

```
In[16]:= {REFLEXIVE[x], ANTISYMMETRIC[x], TRANSITIVE[x], idempotent[x]} /. x → RESTRICT

Out[16]= {False, True, True, True}
```

The breakdown of the reflexive law stems from the fact that its domain is larger than its fixed point set.

```
In[17]:= {domain[RESTRICT], range[RESTRICT], fix[RESTRICT]}

Out[17]= {V, P[cart[V, V]], P[cart[V, V]]}
```

It will now be shown using **AssertTest** that the restriction of **RESTRICT** to relations is a partial order. Sequestering **RESTRICT** from the action of **AssertTest** speeds this up.

```
In[18]:= (REFLEXIVE[composite[x, id[P[cart[V, V]]]]] // AssertTest) /. x → RESTRICT

Out[18]= REFLEXIVE[composite[RESTRICT, id[P[cart[V, V]]]]] == True

In[19]:= REFLEXIVE[composite[RESTRICT, id[P[cart[V, V]]]]] := True

In[20]:= (TRANSITIVE[composite[x, id[P[cart[V, V]]]]] // AssertTest) /. x → RESTRICT

Out[20]= TRANSITIVE[composite[RESTRICT, id[P[cart[V, V]]]]] == True

In[21]:= TRANSITIVE[composite[RESTRICT, id[P[cart[V, V]]]]] := True
```

Combining these lemmas yields the stated result.

```
In[22]:= SubstTest[and, REFLEXIVE[x], ANTISYMMETRIC[x], TRANSITIVE[x],
           x → composite[RESTRICT, id[P[cart[V, V]]]]] // Reverse

Out[22]= PARTIALORDER[composite[RESTRICT, id[P[cart[V, V]]]]] == True

In[23]:= PARTIALORDER[composite[RESTRICT, id[P[cart[V, V]]]]] := True
```

---

a formula for APPLY[VERTSECT[RESTRICT], x]

The relation **RESTRICT** is known to be thin:

```
In[24]:= thin[RESTRICT]

Out[24]= True
```

On account of this, the image under **RESTRICT** of any set is a set:

```
In[25]:= Map[implies[member[x, y], #] &, SubstTest[implies,
           and[thin[z], member[x, V]], member[image[z, x], V], z → RESTRICT]]

Out[25]= or[member[image[COMPOSE, cart[x, P[Id]]], V], not[member[x, y]]] == True

In[26]:= or[member[image[COMPOSE, cart[x_, P[Id]]], V], not[member[x_, y_]]] := True
```

In particular, all its vertical sections are sets:

```
In[27]:= member[image[RESTRICT, singleton[x]], V]

Out[27]= True
```

The following consequence of this will be used to simplify a formula for the application of the function **VERTSECT[RESTRICT]**.

```
In[28]:= equal[union[complement[image[V, singleton[x]]],
           complement[image[V, singleton[RS[x]]]]],
          complement[image[V, singleton[x]]]]

Out[28]= True

In[29]:= union[complement[image[V, singleton[x_]]],
           complement[image[V, singleton[RS[x_]]]]] :=
          complement[image[V, singleton[x]]]
```

One now obtains the following clean formula for the application of the function **VERTSECT[RESTRICT]** to any argument.

```
In[30]:= SubstTest[A, image[z, singleton[x]], z → VERTSECT[RESTRICT]] // Reverse

Out[30]= APPLY[VERTSECT[RESTRICT], x] ==
          union[complement[image[V, singleton[x]]], RS[x]]

In[31]:= APPLY[VERTSECT[RESTRICT], x_] :=
          union[complement[image[V, singleton[x]]], RS[x]]
```

Note that for sets, this formula simplifies further:

```
In[32]:= APPLY[VERTSECT[RESTRICT], setpart[x]]

Out[32]= RS[setpart[x]]
```

In other words, one can regard **VERTSECT[RESTRICT]** as the function which takes any set **x** to **RS[x]**. This can also be established as follows:

```
In[33]:= lambda[x, RS[x]]

Out[33]= VERTSECT[RESTRICT]
```

Since **RS[x]** is a set whenever **x** is a set, the function **lambda[x, RS[x]] = VERTSECT[RESTRICT]** is total:

```
In[34]:= domain[VERTSECT[RESTRICT]]

Out[34]= V
```

This is equivalent to the statement that the relation **RESTRICT** is thin.

variable-free formulation of a characterization of functions

One can use the function **VERTSECT[RESTRICT]** to derive variable-free equations that capture that part of the theory of the class **RS[x]** that applies to the special case that **x** is a set. For example, the formula for restrictions of a function can be written as follows:

```
In[35]:= composite[VERTSECT[RESTRICT], FUNPART] // VSNormality

Out[35]= composite[VERTSECT[RESTRICT], FUNPART] == composite[POWER, FUNPART]

In[36]:= composite[VERTSECT[RESTRICT], FUNPART] := composite[POWER, FUNPART]
```

The theorem that **RS[x] = P[x]** characterizes functions can be written without variables as follows:

```
In[37]:= fix[composite[inverse[POWER], VERTSECT[RESTRICT]]] // Normality

Out[37]= fix[composite[inverse[POWER], VERTSECT[RESTRICT]]] == FUNS

In[38]:= fix[composite[inverse[POWER], VERTSECT[RESTRICT]]] := FUNS
```

Corollary 1.

```
In[39]:= SubstTest[implies, and[subclass[u, v], FUNCTION[v]],
          equal[u, composite[v, id[domain[u]]]],
          {u → intersection[POWER, VERTSECT[RESTRICT]], v → POWER}]

Out[39]= equal[composite[POWER, id[FUNS]],
           intersection[POWER, VERTSECT[RESTRICT]]] == True

In[40]:= intersection[POWER, VERTSECT[RESTRICT]] := composite[POWER, id[FUNS]]
```

Corollary 2.

```
In[41]:= SubstTest[range, intersection[u, v],
           {u -> POWER, v -> VERTSECT[RESTRICT]}] // Reverse

Out[41]= fix[composite[POWER, inverse[VERTSECT[RESTRICT]]]] == image[POWER, FUNS]

In[42]:= fix[composite[POWER, inverse[VERTSECT[RESTRICT]]]] := image[POWER, FUNS]
```

variable-free reformulations of some other facts about RS[x]

The variable-free counterpart of the equation **U[RS[x]] = thinpart[x]** is already known:

```
In[43]:= composite[BIGCUP, VERTSECT[RESTRICT]]

Out[43]= IMAGE[id[cart[V, V]]]
```

The variable-free version of the formula **RS[composite[Id,x]] = RS[x]** can be derived as follows:

```
In[44]:= Map[composite[VERTSECT[#], SINGLETON] &,
          Assoc[RESTRICT, IMAGE[id[cart[V, V]]], inverse[E]]]

Out[44]= composite[VERTSECT[RESTRICT], IMAGE[id[cart[V, V]]]] == VERTSECT[RESTRICT]
```

A less clever derivation of this result is possible, using a standard technique using **VSNormality** and **symdif**. This requires two steps:

```
In[45]:= symdif[composite[VERTSECT[RESTRICT], IMAGE[id[cart[V, V]]]],
           VERTSECT[RESTRICT]] // VSNormality

Out[45]= union[intersection[complement[VERTSECT[RESTRICT]],
            composite[VERTSECT[RESTRICT], IMAGE[id[cart[V, V]]]]],
           intersection[composite[complement[VERTSECT[RESTRICT]],
             IMAGE[id[cart[V, V]]]], VERTSECT[RESTRICT]]] == 0

In[46]:= % /. Equal → SetDelayed

In[47]:= SubstTest[equal, 0, symdif[u, v],
          {u -> composite[VERTSECT[RESTRICT], IMAGE[id[cart[V, V]]]],
           v -> VERTSECT[RESTRICT]}]

Out[47]= True == equal[
           composite[VERTSECT[RESTRICT], IMAGE[id[cart[V, V]]]], VERTSECT[RESTRICT]]

In[48]:= composite[VERTSECT[RESTRICT], IMAGE[id[cart[V, V]]]] := VERTSECT[RESTRICT]
```

reification rule for RS[x]

Reification provides a powerful tool for deriving variable-free counterparts of rewrite rules with variables. A useful reification rule for **RS[x]** is obtained as follows:

```
In[49]:= SubstTest[reify, x, image[IMAGE[cross[Id, f[x]]], y], y → P[Id]]

Out[49]= reify[x, RS[f[x]]] ==
          composite[IMAGE[composite[SWAP, RIF, cross[reify[x, f[x]], Id]]],
           CART, id[cart[V, P[Id]]], inverse[FIRST], SINGLETON]

In[50]:= reify[x_, RS[y_]] :=
          composite[IMAGE[composite[SWAP, RIF, cross[reify[x, y], Id]]],
           CART, id[cart[V, P[Id]]], inverse[FIRST], SINGLETON]
```

In particular, one has

```
In[51]:= reify[x, RS[x]]

Out[51]= RESTRICT
```

---

variable-free results obtained using reification

In this section some examples are presented to illustrate the use of reification to recast results about **RS[x]** as equatons without variables. As a first example, reification is used to eliminate the variable **x** from the rewrite rule for **U[RS[x]]**.

```
In[52]:= SubstTest[reify, x, U[f[x]], f → RS] // Reverse

Out[52]= composite[inverse[E], RESTRICT] == composite[inverse[E], IMAGE[id[cart[V, V]]]]
```

This particular equation can also be independently derived from existing facts about **RESTRICT** as follows:

```
In[53]:= Assoc[inverse[E], inverse[S], RESTRICT] // Reverse

Out[53]= composite[inverse[E], RESTRICT] == composite[inverse[E], IMAGE[id[cart[V, V]]]]

In[54]:= composite[inverse[E], RESTRICT] :=
          composite[inverse[E], IMAGE[id[cart[V, V]]]]
```

The same technique can be used to derive other new properties of **RESTRICT**, for instance,

```
In[55]:= Assoc[complement[inverse[E]], S, RESTRICT] // Reverse

Out[55]= composite[complement[inverse[E]], RESTRICT] == cart[V, V]
```

This result corresponds to the fact that **A[RS[x]] = 0**, as can be seen from the following alternate derivation, using reification:

```
In[56]:= Map[composite[Id, complement[#]] &,
           SubstTest[reify, x, A[f[x]], f → RS]] // Reverse

Out[56]= composite[complement[inverse[E]], RESTRICT] == cart[V, V]

In[57]:= composite[complement[inverse[E]], RESTRICT] := cart[V, V]
```

Aclosure and Uclosure

The facts that **RS[x]** is closed under arbitrary intersections and under arbitrary unions have simple variable-free formulations:

```
In[58]:= Map[VERTSECT, SubstTest[reify, x, Aclosure[f[x]], f → RS]] // Reverse

Out[58]= composite[ACLOSURE, VERTSECT[RESTRICT]] == VERTSECT[RESTRICT]

In[59]:= composite[ACLOSURE, VERTSECT[RESTRICT]] := VERTSECT[RESTRICT]

In[60]:= Map[VERTSECT, SubstTest[reify, x, Uclosure[f[x]], f → RS]] // Reverse

Out[60]= composite[UCLOSURE, VERTSECT[RESTRICT]] == VERTSECT[RESTRICT]

In[61]:= composite[UCLOSURE, VERTSECT[RESTRICT]] := VERTSECT[RESTRICT]
```

ONEONE property of a restriction of VERTSECT[RESTRICT]

One can see that the function **VERTSECT[RESTRICT]** fails to be one-to-one because **RS[x]** for example is equal to **RS[composite[Id,x]]**. It will be shown that the restriction of the function **VERTSECT[RESTRICT]** to the class **P[cart[V, V]]** of all relations is one-to-one. The one-to-one property will be derived by analogy with the following:

```
In[62]:= SubstTest[implies, equal[u, v], equal[U[u], U[v]], {u → RS[x], v → RS[y]}]

Out[62]= or[equal[thinpart[x], thinpart[y]], not[equal[RS[x], RS[y]]]] == True

In[63]:= or[equal[thinpart[x_], thinpart[y_]], not[equal[RS[x_], RS[y_]]]] := True
```

Lemma.

```
In[64]:= composite[inverse[VERTSECT[RESTRICT]], inverse[BIGCUP]] // DoubleInverse

Out[64]= composite[inverse[VERTSECT[RESTRICT]], inverse[BIGCUP]] ==
          inverse[IMAGE[id[cart[V, V]]]]
```

```
In[65]:= composite[inverse[VERTSECT[RESTRICT]], inverse[BIGCUP]] :=
          inverse[IMAGE[id[cart[V, V]]]]

In[66]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → Id, v -> composite[inverse[BIGCUP], BIGCUP],
           w -> inverse[cross[VERTSECT[RESTRICT], VERTSECT[RESTRICT]]]}]

Out[66]= subclass[composite[inverse[VERTSECT[RESTRICT]], VERTSECT[RESTRICT]],
           composite[inverse[IMAGE[id[cart[V, V]]]], IMAGE[id[cart[V, V]]]]] == True

In[67]:= % /. Equal → SetDelayed
```

For the reverse inclusion one needs the following lemma:

```
In[68]:= composite[inverse[IMAGE[id[cart[V, V]]]],
           inverse[VERTSECT[RESTRICT]]] // DoubleInverse

Out[68]= composite[inverse[IMAGE[id[cart[V, V]]]], inverse[VERTSECT[RESTRICT]]] ==
          inverse[VERTSECT[RESTRICT]]

In[69]:= composite[inverse[IMAGE[id[cart[V, V]]]], inverse[VERTSECT[RESTRICT]]] :=
          inverse[VERTSECT[RESTRICT]]

In[70]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → Id, v → composite[inverse[VERTSECT[RESTRICT]], VERTSECT[RESTRICT]],
           w → inverse[cross[IMAGE[id[cart[V, V]]], IMAGE[id[cart[V, V]]]]]}]

Out[70]= subclass[composite[inverse[IMAGE[id[cart[V, V]]]], IMAGE[id[cart[V, V]]]],
           composite[inverse[VERTSECT[RESTRICT]], VERTSECT[RESTRICT]]] == True

In[71]:= % /. Equal → SetDelayed
```

Putting these facts together yields:

```
In[72]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u -> composite[inverse[IMAGE[id[cart[V, V]]]], IMAGE[id[cart[V, V]]]],
           v -> composite[inverse[VERTSECT[RESTRICT]], VERTSECT[RESTRICT]]}]

Out[72]= True == equal[composite[inverse[IMAGE[id[cart[V, V]]]], IMAGE[id[cart[V, V]]]],
           composite[inverse[VERTSECT[RESTRICT]], VERTSECT[RESTRICT]]]

In[73]:= composite[inverse[VERTSECT[RESTRICT]], VERTSECT[RESTRICT]] :=
          composite[inverse[IMAGE[id[cart[V, V]]]], IMAGE[id[cart[V, V]]]]
```

Corollary.

```
In[74]:= SubstTest[subclass, composite[x, inverse[x]], id[y],
           {x -> composite[id[P[cart[V, V]]], inverse[VERTSECT[RESTRICT]]],
            y → P[cart[V, V]]}] // Reverse

Out[74]= FUNCTION[composite[id[P[cart[V, V]]], inverse[VERTSECT[RESTRICT]]]] == True

In[75]:= FUNCTION[composite[id[P[cart[V, V]]], inverse[VERTSECT[RESTRICT]]]] := True
```

a formula for image[inverse[S], RS[x]]

Lemma.

```
In[76]:= Assoc[x, id[domain[VERTSECT[x]]], id[y]]

Out[76]= composite[x, id[intersection[y, domain[VERTSECT[x]]]]] ==
          composite[thinpart[x], id[y]]

In[77]:= composite[x_, id[intersection[y_, domain[VERTSECT[x_]]]]] :=
          composite[thinpart[x], id[y]]
```

If **y** is a subset of **thinpart[x]**, then **y** is contained in the restriction of **thinpart[x]** to **domain[y]**. This is the idea behind the following.

```
In[78]:= implies[member[domain[y], V],
           member[composite[thinpart[x], id[domain[y]]], RS[x]]] // NotNotTest

Out[78]= or[and[member[image[thinpart[x], domain[y]], V],
            member[intersection[domain[x], domain[y], domain[VERTSECT[x]]], V]],
           not[member[domain[y], V]]] == True

In[79]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed

In[80]:= Map[equal[V, class[y, #]] &,
          Map[not, SubstTest[and, implies[and[p1, p2], p3], implies[p1, p4],
             implies[and[p1, p2, p3, p4], p5],
             implies[and[p5, p6], p7], implies[and[p3, p7], p8],
             not[implies[and[p1, p2, p6], p8]], {p1 → member[y, P[thinpart[x]]],
              p2 → equal[w, composite[thinpart[x], id[domain[y]]]], p3 → subclass[y, w],
              p4 → member[domain[y], V], p5 → member[w, RS[x]], p6 → equal[z, RS[x]],
              p7 → member[w, z], p8 -> member[y, image[inverse[S], z]]}]] /.
           {w -> composite[thinpart[x], id[domain[y]]], z → RS[x]}]

Out[80]= subclass[P[thinpart[x]], image[inverse[S], RS[x]]] == True

In[81]:= (% /. x → x_) /. Equal → SetDelayed
```

The reverse inclusion also holds, so one can sharpen this to an equation:

```
In[82]:= SubstTest[and, subclass[u, v], subclass[v, u],
          {u → P[thinpart[x]], v → image[inverse[S], RS[x]]}]

Out[82]= True == equal[image[inverse[S], RS[x]], P[thinpart[x]]]

In[83]:= image[inverse[S], RS[x_]] := P[thinpart[x]]
```

The variable-free equation corresponding to this is:

```
In[84]:= Map[composite[VERTSECT[#], SINGLETON] &,
          Assoc[inverse[S], RESTRICT, inverse[E]]]

Out[84]= composite[IMAGE[inverse[S]], VERTSECT[RESTRICT]] ==
          composite[POWER, IMAGE[id[cart[V, V]]]]

In[85]:= composite[IMAGE[inverse[S]], VERTSECT[RESTRICT]] :=
          composite[POWER, IMAGE[id[cart[V, V]]]]
```

---

some new formulas for RESTRICT

The following old formula for **RESTRICT** is useful for deriving various properties of this relation

```
In[86]:= composite[COMPOSE, id[cart[V, P[Id]]], inverse[FIRST]]

Out[86]= RESTRICT
```

For example, one can use it to derive the following property of **RESTRICT**:

```
In[87]:= Assoc[IMAGE[SECOND], COMPOSE, composite[id[cart[V, P[Id]]], inverse[FIRST]]]

Out[87]= composite[IMAGE[SECOND], RESTRICT] == composite[IMG, inverse[FIRST]]
```

This equation can also be derived a different way using this famous connection between the functions **IMG** and **CART**:

```
In[88]:= composite[IMAGE[rotate[Id]], CART]

Out[88]= IMG
```

The following alternate derivation is based on this idea:

```
In[89]:= Assoc[IMAGE[rotate[Id]], CART, inverse[FIRST]]

Out[89]= composite[IMAGE[SECOND], RESTRICT] == composite[IMG, inverse[FIRST]]

In[90]:= composite[IMAGE[SECOND], RESTRICT] := composite[IMG, inverse[FIRST]]
```

A lemma is needed to get a corresponding result with **IMAGE[FIRST]** in place of **IMAGE[SECOND]**.

```
In[91]:= composite[IMG, id[cart[P[Id], V]], inverse[SECOND]] // VSNormality

Out[91]= composite[IMG, id[cart[P[Id], V]], inverse[SECOND]] == inverse[S]

In[92]:= composite[IMG, id[cart[P[Id], V]], inverse[SECOND]] := inverse[S]

In[93]:= Assoc[IMAGE[FIRST], COMPOSE, composite[id[cart[V, P[Id]]], inverse[FIRST]]]

Out[93]= composite[IMAGE[FIRST], RESTRICT] == composite[inverse[S], IMAGE[FIRST]]

In[94]:= composite[IMAGE[FIRST], RESTRICT] := composite[inverse[S], IMAGE[FIRST]]
```

---

interpreting the formulas

In addition to finding variable-free equations corresponding to properties of **RS[x]**, one can also try to do the reverse, finding properties of **RS[x]** that correspond to identities involving **RESTRICT**. This is less automatic because the variable-free equations only yield results valid when **x** is a set. One needs to go further to obtain completely general results. Nonetheless, the variable-free requations are useful for suggesting what one must do. For example, the interpretation of the **IMAGE[SECOND]** formula found in the preceding section is that the class of all ranges of restrictions of **x** is the range of the function **IMAGE[x]**. To get a general result of this sort, one needs to use a method that does not require **x** to be a set. In this case, the following works:

```
In[95]:= ImageComp[IMAGE[SECOND], IMAGE[cross[Id, x]], P[Id]] // Reverse

Out[95]= image[IMAGE[SECOND], RS[x]] == range[IMAGE[x]]

In[96]:= image[IMAGE[SECOND], RS[x_]] := range[IMAGE[x]]
```

The interpretation of the **IMAGE[FIRST]** formula is that the class of all domains of restrictions of **x** is the power class of the domain of the thinpart of **x**. This fact is established as follows:

```
In[97]:= Map[equal[P[domain[thinpart[x]]], image[#, P[Id]]] &,
          composite[IMAGE[FIRST], IMAGE[cross[Id, x]]] // VSNormality]

Out[97]= equal[image[IMAGE[FIRST], RS[x]],
           P[intersection[domain[x], domain[VERTSECT[x]]]]] == True
```

```
In[101]:=
       image[IMAGE[FIRST], RS[x_]] := P[intersection[domain[x], domain[VERTSECT[x]]]]
```

Recall that the domain of the thin part of **x** is

```
In[99]:= domain[thinpart[x]]

Out[99]= intersection[domain[x], domain[VERTSECT[x]]]
```

When **x** is a function, this formula reduces to

```
In[102]:=
       image[IMAGE[FIRST], P[funpart[x]]]

Out[102]=
       P[domain[funpart[x]]]
```

A similar result holds with the **funpart** wrapper replaced with **thinpart** or **setpart**, but it is currently not known whether the wrapper can be removed altogether. The following inclusion holds, but it is currently not known whether the reverse inclusion holds for arbitrary classes:

```
In[119]:=
       subclass[image[IMAGE[FIRST], P[x]], P[domain[x]]]

Out[119]=
       True
```