the class RS[x] of small restrictions, part 2

Johan G. F. Belinfante
revised 2004 August 9

In[1]:=
```
SetDirectory["i:"]; << goedel60.08b; << tools.m;
```

```
:Package Title: goedel60.08b          2004 August 8 at 2:05 p.m.

It is now:  2004 Aug 9 at 17:1

Loading Simplification Rules

TOOLS.M                    Revised 2004 August 08

weightlimit = 40
```

summary

In this notebook the study of the class **RS[x]** of small restrictions of **x** is continued. A number of results are derived, culminating with the theorem that a class **x** is a function if and only if every subset is a restriction. This result has been the source of several examples in the test suite used to develop the **GOEDEL** program for many years, but the tools to prove it were not available until now.

a (problematic) rule for domain[thinpart[x]]

To simplify expressions involving **thipart[x]**.one often needs to know that the domain of **thinpart[x]** is contained in the domains of **x** and **VERTSECT[x]**, and one sometimes needs a membership rule for the domain of **thinpart[x]**. Instead of adding rules for all these facts, the following single rule is added instead. The orientation of this rule is dictated by the fact that turning it around leads to looping.

In[2]:=
```
domain[thinpart[x]] // Renormality
```

Out[2]=
```
domain[thinpart[x]] == intersection[domain[x], domain[VERTSECT[x]]]
```

In[3]:=
```
domain[thinpart[x_]] := intersection[domain[x], domain[VERTSECT[x]]]
```

a conditional simplification rule for RS[x]

After adding all the rewrite rules derived in part 1 to the **GOEDEL** program, one simple characterization of **RS[x]** mysteriously ceased working, possibly because the order in which rewrite rules are applied could change when the rules are presented in alphabetic order rather than chronological order. The situation is easily fixed by adding a conditional simplification rule, justified by this observation:

```
In[4]:= implies[subclass[thinpart[x], y], equal[intersection[RS[x], P[y]], RS[x]]]

Out[4]= True
```

This is the new rule:

```
In[5]:= intersection[RS[x_], P[y_]] := RS[x] /; subclass[thinpart[x], y]
```

This rule restores the following characterization of **RS[x]** back to its former glory.

```
In[6]:= class[w, equal[w, composite[x, id[domain[w]]]]]

Out[6]= RS[x]
```

While we are at it, here is another result that should have been added in part 1.

```
In[7]:= SubstTest[subclass, intersection[u, v], v,
          {u → P[thinpart[x]], v → invar[composite[id[x], inverse[FIRST], FIRST]]}]

Out[7]= subclass[RS[x], invar[composite[id[x], inverse[FIRST], FIRST]]] == True

In[8]:= subclass[RS[x_], invar[composite[id[x_], inverse[FIRST], FIRST]]] := True
```

function rule

The rule for the class of small restrictions of functions can be derived from the fact that **composite[id[x], inverse[FIRST]]** is one-to-one when **x** is a function, and a conditional rewrite rule for **range[IMAGE[x]]** that holds when **x** is one-to-one. Using these facts, one finds:

```
In[9]:= SubstTest[range, IMAGE[oopart[w]],
          w -> composite[id[funpart[x]], inverse[FIRST]]]

Out[9]= RS[funpart[x]] == P[funpart[x]]
```

The same result can be obtained just as quickly, and with less insight being required, using **Normality**.

```
In[10]:= RS[funpart[x]] // Normality

Out[10]= RS[funpart[x]] == P[funpart[x]]

In[11]:= RS[funpart[x_]] := P[funpart[x]]
```

This rewrite rule says that every subset of a function is a small restriction. This rule in particular applies to constant functions, yielding one of several special rules for restrictions of cartesian products.

```
In[12]:= SubstTest[RS, funpart[w], w → cart[x, singleton[y]]]

Out[12]= RS[cart[x, singleton[y]]] == P[cart[x, singleton[y]]]

In[13]:= RS[cart[x_, singleton[y_]]] := P[cart[x, singleton[y]]]
```

It is unnecessary to add rules for particular cases, such as this one, because one can simply add a conditional rule that applies to all cases. We proceed to do this.

```
In[14]:= SubstTest[implies, and[equal[x, y], equal[RS[y], P[y]]],
          equal[RS[x], P[x]], y → funpart[x]]

Out[14]= or[equal[P[x], RS[x]], not[FUNCTION[x]]] == True

In[15]:= (% /. x → x_) /. Equal → SetDelayed
```

Based on this observation, the following useful conditional rewite rule for functions is justified:

```
In[16]:= RS[x_] := P[x] /; FUNCTION[x]
```

A converse statement will be derived in a later section of this notebook.

restrictions of restrictions

A formula for the small restrictions of a given restriction can be derived:

```
In[17]:= Map[range, composite[IMAGE[cross[Id, x]], IMAGE[id[id[y]]]] // VSNormality]

Out[17]= image[IMAGE[cross[Id, x]], P[id[y]]] == RS[composite[x, id[y]]]

In[18]:= image[IMAGE[cross[Id, x_]], P[id[y_]]] := RS[composite[x, id[y]]]
```

In general, **subvar** is monotone and **invar** is antitone, and their interesection is neither. So one does not expect **RS[x]** to be monotone in general, but for restrictions, there is such a result:

```
In[19]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u → P[id[y]], v → P[Id], w -> IMAGE[cross[Id, x]]}]

Out[19]= subclass[RS[composite[x, id[y]]], RS[x]] == True

In[20]:= subclass[RS[composite[x_, id[y_]]], RS[x_]] := True
```

RS[thinpart[x]] = RS[x]

Since **thinpart[x]** is a restriction of **x**, the result derived in the preceding section holds for it:

```
In[21]:= SubstTest[subclass, RS[composite[x, id[y]]], RS[x], y → domain[VERTSECT[x]]]

Out[21]= subclass[RS[thinpart[x]], RS[x]] == True

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

In fact the small restrictions of **x** are exactly the same as those for **thinpart[x]** as will now be shown.

```
In[23]:= SubstTest[subclass, core[u, v], v,
          {u -> invar[composite[id[x], inverse[FIRST], FIRST]],
           v -> complement[cart[complement[domain[VERTSECT[x]]], V]]}]

Out[23]= subclass[
           domain[core[invar[composite[id[x], inverse[FIRST], FIRST]], complement[
              cart[complement[domain[VERTSECT[x]]], V]]]], domain[VERTSECT[x]]] == True

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

The inclusion in the other direction wil be derived from the fact that **invar** is antitone.

```
In[25]:= SubstTest[implies, subclass[u, v], subclass[invar[v], invar[u]],
          {u → composite[id[thinpart[x]], inverse[FIRST], FIRST],
           v → composite[id[x], inverse[FIRST], FIRST]}]

Out[25]= subclass[invar[composite[id[x], inverse[FIRST], FIRST]],
           invar[composite[id[thinpart[x]], inverse[FIRST], FIRST]]] == True

In[26]:= (% /. x → x_) /. Equal → SetDelayed
```

Note that the power class in the following rule for **RS[thinpart[x]]** is the same that occurs in the corresponding rule for **RS[x]** itself.

```
In[27]:= SubstTest[intersection, invar[composite[id[y], inverse[FIRST], FIRST]],
          P[thinpart[y]], y → thinpart[x]]

Out[27]= intersection[invar[composite[id[thinpart[x]], inverse[FIRST], FIRST]],
          P[thinpart[x]]] == RS[thinpart[x]]

In[28]:= intersection[invar[composite[id[thinpart[x_]], inverse[FIRST], FIRST]],
          P[thinpart[x_]]] := RS[thinpart[x]]
```

The desired inclusion follows:

```
In[29]:= SubstTest[implies, subclass[u, v], subclass[image[w, u], image[w, v]],
          {u -> invar[composite[id[x], inverse[FIRST], FIRST]],
           v -> invar[composite[id[thinpart[x]], inverse[FIRST], FIRST]],
           w → id[P[thinpart[x]]]}]

Out[29]= subclass[RS[x], RS[thinpart[x]]] == True

In[30]:= (% /. x → x_) /. Equal → SetDelayed
```

Combing these two inclusions yields an equation:

```
In[31]:= SubstTest[and, subclass[u, v], subclass[v, u], {u → RS[x], v → RS[thinpart[x]]}]

Out[31]= True == equal[RS[x], RS[thinpart[x]]]

In[32]:= RS[thinpart[x_]] := RS[x]
```

---

restrictions of composites

The result of the preceding section allows one to derive a simple rule for restrictions of composites.

```
In[33]:= Map[image[#, P[Id]] &,
          composite[IMAGE[cross[Id, x]], IMAGE[cross[Id, thinpart[y]]]] // VSNormality]

Out[33]= image[IMAGE[cross[Id, x]], RS[y]] == RS[composite[x, thinpart[y]]]

In[34]:= image[IMAGE[cross[Id, x_]], RS[y_]] := RS[composite[x, thinpart[y]]]
```

In particular, for functions one obtains:

```
In[35]:= SubstTest[image, IMAGE[cross[Id, x]], RS[z], z → funpart[y]]

Out[35]= image[IMAGE[cross[Id, x]], P[funpart[y]]] == RS[composite[x, funpart[y]]]

In[36]:= image[IMAGE[cross[Id, x_]], P[funpart[y_]]] := RS[composite[x, funpart[y]]]
```

---

another formula for restrictions of restrictions

In general the composite of **IMAGE[cross[x,Id]]** and **IMAGE[cross[Id,y]]** need not be equal to **IMAGE[cross[x,y]]**.

```
In[37]:= Map[equal[IMAGE[cross[x, y]], #] &,
          composite[IMAGE[cross[x, Id]], IMAGE[cross[Id, y]]] // VSNormality]

Out[37]= equal[composite[IMAGE[cross[x, Id]], IMAGE[cross[Id, y]]],
            IMAGE[cross[x, y]]] ==
          or[equal[V, domain[VERTSECT[y]]], subclass[domain[VERTSECT[x]], domain[x]]]
```

Even in the special case that x is an identity function, equality does not hold in general:

```
In[38]:= % /. x → id[w]

Out[38]= equal[composite[IMAGE[id[cart[w, V]]], IMAGE[cross[Id, y]]],
            IMAGE[cross[id[w], y]]] == or[equal[V, domain[VERTSECT[y]]], subclass[V, w]]
```

For the special case that **y** is thin, however, one does have a simple rule:

```
In[39]:= composite[IMAGE[cross[x, Id]], IMAGE[cross[Id, thinpart[y]]]] // VSNormality

Out[39]= composite[IMAGE[cross[x, Id]], IMAGE[cross[Id, thinpart[y]]]] ==
          IMAGE[cross[x, thinpart[y]]]

In[40]:= composite[IMAGE[cross[x_, Id]], IMAGE[cross[Id, thinpart[y_]]]] :=
          IMAGE[cross[x, thinpart[y]]]
```

For the special case that x is replaced by an identity function, one obtains:

```
In[41]:= SubstTest[composite, IMAGE[cross[w, Id]],
          IMAGE[cross[Id, thinpart[y]]], w → id[x]]

Out[41]= composite[IMAGE[id[cart[x, V]]], IMAGE[cross[Id, thinpart[y]]]] ==
          IMAGE[cross[id[x], thinpart[y]]]

In[42]:= composite[IMAGE[id[cart[x_, V]]], IMAGE[cross[Id, thinpart[y_]]]] :=
          IMAGE[cross[id[x], thinpart[y]]]
```

Lemma:

```
In[43]:= SubstTest[RS, thinpart[z], z → composite[x, id[y]]]

Out[43]= RS[composite[thinpart[x], id[y]]] == RS[composite[x, id[y]]]

In[44]:= RS[composite[thinpart[x_], id[y_]]] := RS[composite[x, id[y]]]

In[45]:= ImageComp[IMAGE[id[cart[x, V]]],
           IMAGE[cross[Id, thinpart[y]]], P[Id]] // Reverse

Out[45]= image[IMAGE[id[cart[x, V]]], RS[y]] ==
          image[IMAGE[cross[id[x], thinpart[y]]], P[Id]]

In[46]:= image[IMAGE[id[cart[x_, V]]], RS[y_]] := RS[composite[y, id[x]]]
```

FUNCTION characterization

A proof that functions are characterized by **P[x] = RS[x]** will now be presented. The starting point is this special formula for cartesian products whose domains are singletons:

```
In[47]:= SubstTest[image, IMAGE[cross[Id, z]],
           P[Id], z -> cart[singleton[x], y]] // Reverse

Out[47]= RS[cart[singleton[x], y]] == pairset[0, cart[singleton[x], y]]

In[48]:= RS[cart[singleton[x_], y_]] := pairset[0, cart[singleton[x], y]]
```

Here is a companion formula for restrictions of restrictions:

```
In[49]:= ImageComp[IMAGE[cross[Id, x]], IMAGE[id[cart[y, V]]], P[Id]]

Out[49]= image[IMAGE[cross[id[y], x]], P[Id]] == RS[composite[x, id[y]]]

In[50]:= image[IMAGE[cross[id[y_], x_]], P[Id]] := RS[composite[x, id[y]]]
```

A cartesian product is a singleton if and only if it is the cartesian product of two singletons:

```
In[51]:= member[cart[x, y], range[SINGLETON]] // AssertTest

Out[51]= member[cart[x, y], range[SINGLETON]] ==
          and[member[x, range[SINGLETON]], member[y, range[SINGLETON]]]

In[52]:= member[cart[x_, y_], range[SINGLETON]] :=
          and[member[x, range[SINGLETON]], member[y, range[SINGLETON]]]
```

This result is applied as follows:

```
In[53]:= SubstTest[implies, equal[u, v], equal[image[w, u], image[w, v]],
          {u → P[x], v → RS[x], w -> IMAGE[id[cart[singleton[y], V]]]}]

Out[53]= or[member[y, domain[funpart[x]]],
           not[equal[P[x], RS[x]]], not[member[y, domain[x]]]] == True

In[54]:= (% /. {x → x_, y → y_}) /. Equal → SetDelayed
```

The next step is to eliminate the variable **y** in the standard way.

```
In[55]:= Map[equal[V, #] &,
           SubstTest[class, y, or[member[y, t], not[equal[u, v]], not[member[y, w]]],
            {t -> domain[funpart[x]], u -> P[x], v → RS[x], w → domain[x]}]] // Reverse

Out[55]= or[FUNCTION[composite[Id, x]], not[subclass[P[x], RS[x]]]] == True

In[56]:= (% /. x → x_) /. Equal → SetDelayed
```

One also needs to know that if every subclass of a class is a restriction, then that class must be a relaton:

```
In[57]:= Map[or[#, subclass[x, cart[V, V]]] &,
          SubstTest[implies, equal[u, v], equal[U[u], U[v]], {u → P[x], v → RS[x]}]]

Out[57]= or[not[equal[P[x], RS[x]]], subclass[x, cart[V, V]]] == True

In[58]:= (% /. x → x_) /. Equal → SetDelayed
```

It just remains to tie up all the loose ends:

```
In[60]:= Map[not, SubstTest[and, implies[p1, p2], implies[p1, p3],
           implies[p3, p4], implies[and[p2, p4], p5], not[implies[p1, p5]],
           {p1 → equal[P[x], RS[x]], p2 → subclass[x, cart[V, V]],
            p3 → subclass[P[x], RS[x]], p4 → FUNCTION[composite[Id, x]],
            p5 → FUNCTION[x]}]]

Out[60]= or[FUNCTION[x], not[equal[P[x], RS[x]]]] == True

In[61]:= (% /. x → x_) /. Equal → SetDelayed
```

The converse also holds, so one can add the following rewrite rule:

```
In[62]:= equiv[equal[P[x], RS[x]], FUNCTION[x]]

Out[62]= True

In[63]:= equal[P[x_], RS[x_]] := FUNCTION[x]
```